Optifol Source Code Listing

Oliver Dixon

Document Generated: 19th May 2026

Source Last Modified: 11th April 2026

Abstract

The following document is a typeset listing of the entire *Optifol* code base as of 11th April 2026. It corresponds with the Git commit hash `a6a52a2e300767c384f3843f35872474a18f1c6d` on branch `typeset`. The repository is available to browse online at `https://github.com/oliverdixon/OptiFOL-Software/tree/typeset`. The LaTeX dissertation describing the planning, development, and evaluation of the software is available at `https://github.com/oliverdixon/OptiFOL-Documentation`. All additional materials, such as compiled documents, are available on my academic website at `https://www-users.york.ac.uk/~od641/l6-project/`. Please direct any queries to Oliver Dixon <od641@york.ac.uk>.

Remarks

This document is not intended to reflect a finish product; rather an archive of the source at the point of submission for the Level 6 dissertation. Given that not all functional requirements were met in the timeline, some code is annotated with `TODO` markers, and Doxygen documentation is occasionally incomplete.

From the front-matter onwards, the page margins shrink from 2cm to 0.5cm to accommodate the 110-column source listings[1]. As some printers may not be able to handle such large text areas, printing a couple of test pages prior to printing the entire document is recommended!

The *File Index* structure below corresponds to the directory hierarchy of the *Optifol* code base: the "core" files are detained in `Source`, and support files can be found in `Resources`. Please note the following exclusions:

- A small number of resources are binary files that are not practical to display in this document.
- Some generated sources (e.g. the lexers and parsers produced by Flex and Bison) are not included because they are long, unwieldy, and serve no expository purpose beyond their source counterparts.

The script and LaTeX template used to create this document can be found online at `https://github.com/oliverdixon/OptiFOL-Software/tree/typeset/Typeset`. Doxygen HTML documentation of the sources can be found at `https://www-users.york.ac.uk/~od641/l6-project/doxygen`.

File Index

1 Source **1**
1.1 CompositeSerialisationHelpers.hpp 1
1.2 IHashable.hpp 2
1.3 ISerialisable.hpp 5
1.4 Logging.cpp 7
1.5 Logging.hpp 7
1.6 Optifol.hpp 8
1.7 Exceptions 13
1.7.1 ParseError.hpp 13
1.7.2 SemanticException.hpp 13
1.8 GUI 14

[1] The software was initially developed with 120-column files. The sources have been automatically reset to 110 columns using the `clang-format` utility, configured by the `.clang-format` directive file provided herein. This process may have caused very occasional quirks in formatting. The original sources can be found via version control on the `master` branch.

1.8.1 Application.cpp 14
1.8.2 Application.hpp 15
1.8.3 ContextButtonCorrespondence.cpp 16
1.8.4 ContextButtonCorrespondence.hpp 17
1.8.5 GTKHelpers.hpp 18
1.8.6 GUIDriver.cpp 22
1.8.7 IWindowArea.hpp 22
1.8.8 MainWindow.cpp 23
1.8.9 MainWindow.hpp 24
1.8.10 ProcessExecutor.cpp 25
1.8.11 ProcessExecutor.hpp 26
1.8.12 StreamingProcessExecutor.cpp 27
1.8.13 StreamingProcessExecutor.hpp 28
1.8.14 AnalysisArea 30
1.8.14.1 AnalysisArea.cpp 30
1.8.14.2 AnalysisArea.hpp 33
1.8.14.3 AnalysisAreaNewAnalysisGroupPopover.cpp 35
1.8.14.4 AnalysisAreaNewAnalysisGroupPopover.hpp 36
1.8.14.5 AnalysisQueryCanvas.cpp 37
1.8.14.6 AnalysisQueryCanvas.hpp 41
1.8.14.7 AnalysisQuery.cpp 43
1.8.14.8 AnalysisQuery.hpp 44
1.8.14.9 CanvasSupport.hpp 45
1.8.15 ProjectHierarchyPane 48
1.8.15.1 ProjectHierarchyPane.cpp 48
1.8.15.2 ProjectHierarchyPane.hpp 52
1.8.16 ReportsArea 54
1.8.16.1 ReportsArea.cpp 54
1.8.16.2 ReportsAreaGenerateLaTeXPopover.cpp 54
1.8.16.3 ReportsAreaGenerateLaTeXPopover.hpp 56
1.8.16.4 ReportsArea.hpp 59
1.8.17 RequirementsIndexArea 60
1.8.17.1 IndexDeleteRequirementPopover.cpp 60
1.8.17.2 IndexDeleteRequirementPopover.hpp 61
1.8.17.3 IndexDuplicateRequirementPopover.cpp 62
1.8.17.4 IndexDuplicateRequirementPopover.hpp 63
1.8.17.5 IndexNewRequirementPopover.cpp 65
1.8.17.6 IndexNewRequirementPopover.hpp 66
1.8.17.7 ManageTestsPopover.cpp 68
1.8.17.8 ManageTestsPopover.hpp 72
1.8.17.9 RequirementsIndexArea.cpp 74
1.8.17.10 RequirementsIndexArea.hpp 77
1.8.18 TestingArea 79
1.8.18.1 TestingArea.cpp 79
1.8.18.2 TestingArea.hpp 82
1.8.18.3 TestingCopyMovePopoverBase.cpp 85
1.8.18.4 TestingCopyMovePopoverBase.hpp 86
1.8.18.5 TestingCopyToTestGroupPopover.cpp 88
1.8.18.6 TestingCopyToTestGroupPopover.hpp 88
1.8.18.7 TestingDeleteTestGroupPopover.cpp 90
1.8.18.8 TestingDeleteTestGroupPopover.hpp 91
1.8.18.9 TestingFailedView.cpp 92
1.8.18.10 TestingFailedView.hpp 94
1.8.18.11 TestingMoveToTestGroupPopover.cpp 95
1.8.18.12 TestingMoveToTestGroupPopover.hpp 96
1.8.18.13 TestingNewTestGroupPopover.cpp 98
1.8.18.14 TestingNewTestGroupPopover.hpp 99
1.8.18.15 TestingRenameTestGroupPopover.cpp 100
1.8.18.16 TestingRenameTestGroupPopover.hpp 101
1.8.18.17 TestingRunTestsPopover.cpp 102
1.8.18.18 TestingRunTestsPopover.hpp 103

1.9 Inference 105
1.9.1 ExpressionFactory.cpp 105
1.9.2 ExpressionFactory.hpp 107
1.9.3 Feature.hpp 108
1.9.4 FVIKnowledgeBase.cpp 110
1.9.5 FVIKnowledgeBase.hpp 114
1.9.6 ProofTreeNode.hpp 115
1.9.7 Prover.cpp 116
1.9.8 Prover.hpp 121
1.9.9 QueryResult.hpp 124
1.9.10 Resolvent.cpp 124
1.9.11 Resolvent.hpp 125
1.9.12 ResolventQueue.cpp 126
1.9.13 ResolventQueue.hpp 128
1.9.14 Unifier.cpp 129
1.9.15 Unifier.hpp 130
1.10 Interpreter 131
1.10.1 FOLLexer.hpp 131
1.10.2 FOLLexer.l 131
1.10.3 FOLParser.hpp 132
1.10.4 FOLParser.y 133
1.11 IR 135
1.11.1 SymbolRepository.cpp 135
1.11.2 SymbolRepository.hpp 136
1.11.3 MutableVariants 139
1.11.3.1 OwningBuildable.hpp 139
1.11.3.2 Sentences 139
1.11.3.2.1 IMutableSentence.hpp 139
1.11.3.2.2 MutableBinaryConnected.cpp 141
1.11.3.2.3 MutableBinaryConnected.hpp 142
1.11.3.2.4 MutablePredicate.cpp 144
1.11.3.2.5 MutablePredicate.hpp 146
1.11.3.2.6 MutableQuantified.cpp 147
1.11.3.2.7 MutableQuantified.hpp 150
1.11.3.2.8 MutableSentenceRoot.cpp 152
1.11.3.2.9 MutableSentenceRoot.hpp 153
1.11.3.3 Terms 154
1.11.3.3.1 IMutableTerm.hpp 154
1.11.3.3.2 MutableFunction.cpp 155
1.11.3.3.3 MutableFunction.hpp 157
1.11.3.3.4 MutableSkolemFunction.cpp 158
1.11.3.3.5 MutableSkolemFunction.hpp 158
1.11.3.3.6 MutableVariable.cpp 159
1.11.3.3.7 MutableVariable.hpp 160
1.11.4 Sentences 161
1.11.4.1 BinaryConnected.cpp 161
1.11.4.2 BinaryConnected.hpp 163
1.11.4.3 Clause.cpp 164
1.11.4.4 Clause.hpp 167
1.11.4.5 IProcessedSentence.hpp 169
1.11.4.6 ISentence.hpp 170
1.11.4.7 Literal.cpp 171
1.11.4.8 Literal.hpp 173
1.11.4.9 SentenceRoot.cpp 174
1.11.4.10 SentenceRoot.hpp 176
1.11.5 Terms 176
1.11.5.1 Function.cpp 176
1.11.5.2 Function.hpp 178
1.11.5.3 IProcessedTerm.hpp 179
1.11.5.4 ITerm.hpp 181
1.11.5.5 SkolemFunction.cpp 182

1.11.5.6 SkolemFunction.hpp 183
1.11.5.7 Variable.cpp 183
1.11.5.8 Variable.hpp 185
1.12 Reporting 186
1.12.1 IReportGenerator.hpp 186
1.12.2 LaTeXReportGenerator.cpp 187
1.12.3 LaTeXReportGenerator.hpp 189
1.13 Storage 190
1.13.1 AnalysisGroup.cpp 190
1.13.2 AnalysisGroup.hpp 191
1.13.3 ObjectGroup.hpp 192
1.13.4 Project.cpp 195
1.13.5 Project.hpp 196
1.13.6 Requirement.cpp 196
1.13.7 Requirement.hpp 200
1.13.8 StorageObjectBase.cpp 203
1.13.9 StorageObjectBase.hpp 205
1.13.10 Subsystem.cpp 207
1.13.11 Subsystem.hpp 210
1.13.12 TreeNode.cpp 212
1.13.13 TreeNode.hpp 212
1.14 Tests 213
1.14.1 BidirectionalUnificationTest.cpp 213
1.14.2 CNFNormalisationTest.cpp 216
1.14.3 FeatureVectorIndexTest.cpp 221
1.14.4 FOLParserTest.cpp 222
1.14.5 GoogleTestSupport.hpp 224
1.14.6 LiteralOrganisationTest.cpp 224
1.14.7 RepositoryBuildingTest.cpp 227
1.14.8 ResolutionTest.cpp 231
1.14.9 UnidirectionalUnificationTest.cpp 234
1.15 UserTesting 236
1.15.1 Discovery 236
1.15.1.1 DiscoveryTestExecutable.cpp 236
1.15.1.2 DiscoveryTestExecutable.hpp 236
1.15.1.3 DiscoveryTestFixture.cpp 238
1.15.1.4 DiscoveryTestFixture.hpp 239
1.15.1.5 GoogleTestDiscoveryExecutable.cpp 239
1.15.1.6 GoogleTestDiscoveryExecutable.hpp 241
1.15.1.7 TestSpecificationEntry.cpp 242
1.15.1.8 TestSpecificationEntry.hpp 243
1.15.2 Execution 245
1.15.2.1 PartialTestResult.cpp 245
1.15.2.2 PartialTestResult.hpp 246
1.15.2.3 TestExecutable.cpp 246
1.15.2.4 TestExecutable.hpp 247
1.15.2.5 TestResult.cpp 248
1.15.2.6 TestResult.hpp 249
1.15.2.7 PayloadManagement 251
1.15.2.7.1 GoogleTestLexer.hpp 251
1.15.2.7.2 GoogleTestLexer.l 251
1.15.2.7.3 GoogleTestListener.cpp 252
1.15.2.7.4 GoogleTestListener.hpp 254
1.15.2.7.5 GoogleTestParser.hpp 255
1.15.2.7.6 GoogleTestParser.y 256
1.15.2.7.7 TestListenerBase.cpp 259
1.15.2.7.8 TestListenerBase.hpp 260
1.15.3 Modelling 261
1.15.3.1 ExecutionGroup.cpp 261
1.15.3.2 ExecutionGroup.hpp 261
1.15.3.3 GoogleExecutionGroup.cpp 262

1.15.3.4 GoogleExecutionGroup.hpp 264
1.15.3.5 ITestModelNode.hpp 265
1.15.3.6 Test.cpp 266
1.15.3.7 TestGroup.cpp 268
1.15.3.8 TestGroup.hpp 271
1.15.3.9 Test.hpp 274
1.16 Visitors 275
1.16.1 MutableTargets 275
1.16.1.1 RepositoryBuildingVisitor.cpp 275
1.16.1.2 RepositoryBuildingVisitor.hpp 277
1.16.1.3 Observers 279
1.16.1.3.1 IObservingNodeVisitor.hpp 279
1.16.1.3.2 LaTeXSerialisationVisitor.cpp 280
1.16.1.3.3 LaTeXSerialisationVisitor.hpp 282
1.16.1.3.4 TextSerialiserVisitor.cpp 282
1.16.1.3.5 TextSerialiserVisitor.hpp 284
1.16.1.4 Sentences 284
1.16.1.4.1 MutatingSentenceVisitorBase.cpp 284
1.16.1.4.2 MutatingSentenceVisitorBase.hpp 285
1.16.1.4.3 CNFNormalisers 286
1.16.1.4.3.1 DisjunctionDistributionVisitor.cpp 286
1.16.1.4.3.2 DisjunctionDistributionVisitor.hpp 288
1.16.1.4.3.3 DMLVisitor.cpp 289
1.16.1.4.3.4 DMLVisitor.hpp 290
1.16.1.4.3.5 ImplicationEliminationVisitor.cpp 291
1.16.1.4.3.6 ImplicationEliminationVisitor.hpp 292
1.16.1.4.3.7 QuantifierExtractingVisitor.cpp 292
1.16.1.4.3.8 QuantifierExtractingVisitor.hpp 294
1.16.1.4.3.9 SkolemIntroducingVisitor.cpp 297
1.16.1.4.3.10 SkolemIntroducingVisitor.hpp 299
1.16.1.4.3.11 SymbolStandardisingVisitor.cpp 300
1.16.1.4.3.12 SymbolStandardisingVisitor.hpp 301
1.16.1.4.3.13 UniversalEliminationVisitor.cpp 303
1.16.1.4.3.14 UniversalEliminationVisitor.hpp 304
1.16.1.5 Terms 304
1.16.1.5.1 MutatingTermVisitorBase.cpp 304
1.16.1.5.2 MutatingTermVisitorBase.hpp 305
1.16.1.5.3 ScopedTermResolutionVisitor.cpp 306
1.16.1.5.4 ScopedTermResolutionVisitor.hpp 306
1.16.1.5.5 TermResolutionVisitor.cpp 307
1.16.1.5.6 TermResolutionVisitor.hpp 307
1.16.2 RegularTargets 308
1.16.2.1 FeatureBuildingVisitor.cpp 308
1.16.2.2 FeatureBuildingVisitor.hpp 310
1.16.2.3 IObservingBinaryVisitor.hpp 311
1.16.2.4 Unification 311
1.16.2.4.1 BidirectionalUnificationVisitor.cpp 311
1.16.2.4.2 BidirectionalUnificationVisitor.hpp 312
1.16.2.4.3 UnificationApplicationVisitor.cpp 314
1.16.2.4.4 UnificationApplicationVisitor.hpp 315
1.16.2.4.5 UnificationVisitor.cpp 316
1.16.2.4.6 UnificationVisitor.hpp 318

2 CMakeLists.txt 320

3 CMakePresets.json 327

4 UI **327**
4.1 Application.ui 327
4.2 MainWindow 329
4.2.1 MainWindow.ui.in 329
4.2.2 replacements.txt 331
4.2.3 styles.css 332
4.2.4 AnalysisArea 332
4.2.4.1 AnalysisView.ui 332
4.2.4.2 DeleteAnalysisGroupPopover.ui 335
4.2.4.3 EditAnalysisGroupPopover.ui 336
4.2.4.4 NewAnalysisGroupPopover.ui 337
4.2.5 ProjectPane 338
4.2.5.1 DeleteStructurePopover.ui 338
4.2.5.2 EditStructurePopover.ui 339
4.2.5.3 NewProjectPopover.ui 340
4.2.5.4 NewSubsystemPopover.ui 341
4.2.5.5 ProjectPane.ui 342
4.2.6 ReportsArea 343
4.2.6.1 GenerateLaTeXPopover.ui 343
4.2.6.2 ReportsArea.ui 345
4.2.7 RequirementsIndexArea 346
4.2.7.1 DeleteRequirementPopover.ui 346
4.2.7.2 DuplicateRequirementPopover.ui 347
4.2.7.3 EditRequirementPopover.ui 348
4.2.7.4 ManageTestsPopover.ui 351
4.2.7.5 NewRequirementPopover.ui 352
4.2.7.6 RequirementsIndexView.ui 354
4.2.8 TestingArea 356
4.2.8.1 CopyToTestGroupPopover.ui 356
4.2.8.2 DeleteTestGroupPopover.ui 357
4.2.8.3 MoveToTestGroupPopover.ui 358
4.2.8.4 NewTestGroupPopover.ui 359
4.2.8.5 RenameTestGroupPopover.ui 360
4.2.8.6 RunTests.ui 361
4.2.8.7 TestingView.ui 362

5 vcpkg.json **365**

6 .clang-format **365**

1 Source

1.1 CompositeSerialisationHelpers.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Concept definitions for composite iterable-serialisable types
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#ifndef COMPOSITESERIALISABLECONCEPTS_HPP
#define COMPOSITESERIALISABLECONCEPTS_HPP

#include <iostream>
#include <iterator>

namespace optifol
{

/**
 * @concept SerialisableIterator
 * @brief Represents a bidirectional iterator to a de-referencable type with a @ref std::ostream serialisation
 *  function.
 * @details A SerialisableIterator satsifies the following constraints:
 *  <ul>
 *      <li>Compliant with @ref std::bidirectional_iterator;</li>
 *      <li>Be a constant iterator; and</li>
 *      <li>Iterates over a pointer-like type that can be dereferenced to a type with a serialiser member
 *          function.</li>
 *  </ul>
 */
template<typename Candidate>
concept SerialisableIterator = std::bidirectional_iterator<Candidate> &&
        std::is_const_v<std::remove_reference_t<decltype(*std::declval<Candidate>())>> &&
        requires(std::ostream &ostream, const Candidate &candidate) {
            { (*candidate)->serialise(ostream) } -> std::convertible_to<std::ostream &>;
        };

/**
 * @concept StringifiableIterator
 * @brief Represents a bidirectional iterator to a de-referencable type with a @ref std::string stringifier
 *  function.
 * @details A SerialisableIterator satsifies the following constraints:
 *  <ul>
 *      <li>Compliant with @ref std::bidirectional_iterator;</li>
 *      <li>Be a constant iterator; and</li>
 *      <li>Iterates over a pointer-like type that can be dereferenced to a type with a stringifier member
 *          function.</li>
 *  </ul>
 */
template<typename Candidate>
concept StringifiableIterator = std::bidirectional_iterator<Candidate> &&
        std::is_const_v<std::remove_reference_t<decltype(*std::declval<Candidate>())>> &&
        requires(std::ostream &ostream, const Candidate &candidate) {
            { (*candidate)->to_string() } -> std::convertible_to<std::string>;
        };

class CompositeSerialisationHelpers
{
public:
    /**
     * @brief Serialise a formatted symbol string, including all arguments, to the specified output stream.
     * @tparam Iterator @ref SerialisableIterator over the argument collection
     * @param ostream Destination output stream
     * @param display_name Symbol display name
     * @param arguments_begin Beginning iterator of the argument collection
     * @param arguments_end Ending iterator of the argument collection (one past last item)
     * @param is_negative_polarity Is the symbol instantiated with a negative polarity?
     * @return Populated destination output stream
     */
    template<SerialisableIterator Iterator>
    static std::ostream &stream_serialise(std::ostream &ostream, const std::string_view display_name,
```

```
            const Iterator arguments_begin, Iterator arguments_end, const bool is_negative_polarity = false)
    {
        if (is_negative_polarity)
            ostream << '~';

        ostream << display_name;

        if (arguments_begin == arguments_end)
            return ostream;

        ostream << '(';

        if (arguments_begin < arguments_end) {
            --arguments_end;

            for (auto argument_it = arguments_begin; argument_it != arguments_end; ++argument_it) {
                (*argument_it)->serialise(ostream);
                ostream << ", ";
            }

            (*arguments_end)->serialise(ostream);
        }

        return ostream << ')';
    }

    /**
     * @brief Serialise a formatted symbol string, including all arguments, into a dynamic @ref std::string
     *  buffer.
     * @tparam Iterator @ref StringifableIterator over the argument collection
     * @param display_name Symbol display name
     * @param arguments_begin Beginning iterator of the argument collection
     * @param arguments_end Ending iterator of the argument collection (one past last item)
     * @param is_negative_polarity Is the symbol instantiated with a negative polarity?
     * @return Populated string buffer
     */
    template<StringifiableIterator Iterator>
    static std::string string_serialise(std::string display_name, const Iterator arguments_begin,
            Iterator arguments_end, const bool is_negative_polarity = false)
    {
        std::string result;

        if (is_negative_polarity)
            result = '~';

        if (arguments_begin == arguments_end)
            return result + display_name;

        result += std::move(display_name) + '(';

        if (arguments_begin < arguments_end) {
            --arguments_end;

            for (auto argument_it = arguments_begin; argument_it != arguments_end; ++argument_it)
                result += (*argument_it)->to_string() + ", ";

            result += (*arguments_end)->to_string();
        }

        return result + ')';
    }
};

} // namespace optifol

#endif
```

1.2 IHashable.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the IHashable interface, and some supporting specialisations.
 * @author Oliver Dixon
 * @date 2025-06-09
```

```
 * @version Development
 */

#ifndef IHASHABLE_HPP
#define IHASHABLE_HPP

#include <chrono>
#include <cstddef>
#include <memory>

#include "IHashable.hpp"

namespace optifol
{

class IHashable;

template<typename Candidate>
concept HashableIterator = std::bidirectional_iterator<Candidate> &&
        std::is_const_v<std::remove_reference_t<decltype(*std::declval<Candidate>())>> &&
        requires(const Candidate &candidate) {
            { (*candidate)->hash() } -> std::convertible_to<std::size_t>;
        };

/**
 * @class IHashable
 * @brief An IHashable class can be hashed, such that a (relatively) content-dependent numerical hashcode can
 * be generated for any instance. Hashcodes are useful for implementing equality functor, but IHashable makes
 * no guarantees on the collision properties of the generated hashcodes.
 */
class IHashable
{
public:
    /**
     * @brief Destruct an IHashable instance
     */
    virtual ~IHashable() = default;

    /**
     * @brief Produce a hash value of the IHashable object
     * @return A hash value for the object
     */
    [[nodiscard]] virtual std::size_t hash() const noexcept = 0;

    /**
     * @brief Combine two hashes using sensible constants, inspired by boost::hash_combine.
     * @param lhs The LHS hash value
     * @param rhs The RHS hash value
     * @return The LHS-RHS combined hash value
     */
    static std::size_t hash_combine(std::size_t lhs, const std::size_t rhs)
    {
        if constexpr (sizeof(std::size_t) >= 8)
            // For 64-bit+ platforms, use the expansion of pi as the constant; it is odd at 64 bits.
            lhs ^= rhs + 0x517cc1b727220a95 + (lhs << 6) + (lhs >> 2);
        else
            // Otherwise, use the inverse of the golden ratio as a 32-bit fixed point fraction.
            // ReSharper disable once CppDFAUnreachableCode
            lhs ^= rhs + 0x9e3779b9 + (lhs << 6) + (lhs >> 2);

        return lhs;
    }

    IHashable(const IHashable &) = default;
    IHashable &operator=(const IHashable &) = default;
    IHashable(IHashable &&) = default;
    IHashable &operator=(IHashable &&) = default;

protected:
    IHashable() = default;

    /**
     * @brief Commutatively combine two hashes using sensible constants, inspired by boost::hash_combine.
     * @param lhs The LHS hash value
     * @param rhs The RHS hash value
     * @return The LHS-RHS combined hash value such that LHS-RHS hash equals the equivalent RHS-LHS hash
     *  (property of commutativity).
     */
    static std::size_t hash_combine_commutative(std::size_t lhs, std::size_t rhs)
    {
        /*
```

```
         * The choice of operator> to collapse the hash operands into a commutative pair is arbitrary. We just
         * need something reliable and universally defined on std::size_t.
         */
        if (lhs > rhs)
            std::swap(lhs, rhs);

        return hash_combine(lhs, rhs);
    }

    /**
     * @brief Mutate a hash according the polarity of the node instantiation being hashed
     * @param hash The produced hash for the unsigned node (i.e., the node without a polarity)
     * @param is_negative Does the instantiation have a negative sign?
     * @return If positive, the original hash. If negative, a mutated hash to reflect the difference in
     * polarity.
     */
    static std::size_t hash_polarity(const std::size_t hash, const bool is_negative)
    {
        // In the negative case, use MurmurHash3 as the XOR constant, and shift as usual.
        return is_negative ? hash ^ 0x85ebca6b + (hash << 6) + (hash >> 2) : hash;
    }

    /**
     * @brief Produce a hash value of a composite IR IHashable node by iterative application of
     *  @ref hash_combine(std::size_t, std::size_t).
     * @tparam Iterator @ref HashableIterator over the composed (e.g. argument) collection
     * @param symbol_name Symbol display name
     * @param composite_begin Beginning iterator of the composed collection
     * @param composite_end Ending iterator of the composed collection (one past last item)
     * @param is_negative_polarity Is the symbol instantiated with a negative polarity?
     * @return Combined hash value unique over the symbol name and all composed arguments
     */
    template<HashableIterator Iterator>
    static std::size_t composite_hash(const std::string &symbol_name, const Iterator composite_begin,
            const Iterator composite_end, const bool is_negative_polarity = false)
    {
        std::size_t hashcode = std::hash<std::string>{}(symbol_name);

        for (auto composite_it = composite_begin; composite_it != composite_end; ++composite_it)
            hashcode = hash_combine(hashcode, (*composite_it)->hash());

        return hash_polarity(hashcode, is_negative_polarity);
    }
};

/**
 * @class StringHash
 * @brief Hashing functor helper for transparent/heterogeneous lookup on string-like containers.
 */
struct StringHash
{
    using is_transparent = void;
    using hash_type = std::hash<std::string_view>;

    auto operator()(const char *const str) const
    {
        return hash_type{}(str);
    }
    auto operator()(const std::string_view str) const
    {
        return hash_type{}(str);
    }
    auto operator()(const std::string &str) const
    {
        return hash_type{}(str);
    }
};

struct PairHash
{
    template<typename LHS, typename RHS>
        requires(std::derived_from<LHS, IHashable> and std::derived_from<RHS, IHashable>)
    auto operator()(const std::pair<LHS, RHS> &pair) const
    {
        return IHashable::hash_combine(pair.first.hash(), pair.second.hash());
    }
};

} // namespace optifol

// ReSharper disable once CppDoxygenUnresolvedReference
```

```
namespace std
{

/**
 * @class hash<Type>
 * @brief Standard hasher implementation for Optifol's IHashable derived classes
 * @tparam Type The IHashable type to hash
 */
template<typename Type>
    requires derived_from<Type, optifol::IHashable>
struct hash<Type> // NOLINT(*-dcl58-cpp) Specialising std::hash does not result in UB.
{
    using is_transparent = void;

    /**
     * @brief Execute the hash functor to produce a hashcode of the object
     * @param hashable The hashable object for which a hashcode should be generated
     * @return The hashcode of the hashable object
     */
    std::size_t operator()(const Type &hashable) const noexcept
    {
        return hashable.hash();
    }

    /**
     * @brief Execute the hash functor to produce a hashcode of the object contained within the ref-counted
     *  pointer
     * @param shared_hashable The ref-counted pointer containing the hashable object for which a hashcode
     *  should be generated
     * @return The hashcode of the hashable object detained within the ref-counted pointer
     */
    size_t operator()(const shared_ptr<Type> &shared_hashable) const
    {
        return shared_hashable->hash();
    }

    /**
     * @brief Execute the hash functor to produce a hashcode of the object contained within the unique pointer
     * @param unique_hashable The unique pointer containing the hashable object for which a hashcode should be
     *  generated
     * @return The hashcode of the hashable object detained within the unique pointer
     */
    size_t operator()(const unique_ptr<Type> &unique_hashable) const
    {
        return unique_hashable->hash();
    }

    size_t operator()(const Type *ptr_hashable) const
    {
        return ptr_hashable->hash();
    }
};

#if __cpp_lib_chrono < 202306L

// ReSharper disable once CppDoxygenUnresolvedReference

/**
 * @class hash<std::chrono::system_clock::time_point>
 * @brief Standard hasher specialisation for the system clock, only required prior to C++26.
 */
template<>
struct hash<chrono::system_clock::time_point>
{
    std::size_t operator()(const chrono::system_clock::time_point &time) const noexcept;
};

#endif

} // namespace std

#endif
```

1.3 ISerialisable.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
```

```
 */

/**
 * @file
 * @brief Class specification for the ISerialisable interface
 * @author Oliver Dixon
 * @date 2026-01-25
 * @version Development
 */

#ifndef OPTIFOL_ISERIALISABLE_HPP
#define OPTIFOL_ISERIALISABLE_HPP

#include <format>
#include <sstream>

namespace optifol
{

/**
 * @class ISerialisable
 * @brief An interface for objects that may be serialised to a @ref std::ostream
 */
class ISerialisable
{
public:
    /**
     * @brief Destruct the serialisable object
     */
    virtual ~ISerialisable() = default;

    /**
     * @brief Serialise the object to the given output stream
     * @param ostream The destination output stream
     * @return The populated output stream
     */
    virtual std::ostream &serialise(std::ostream &ostream) const = 0;

    /**
     * @brief Operator overload to serialise the object to the given output stream
     * @param ostream The destination output stream
     * @param object The object to serialise
     * @return The populated output stream
     */
    friend std::ostream &operator<<(std::ostream &ostream, const ISerialisable &object)
    {
        return object.serialise(ostream);
    }

    ISerialisable(const ISerialisable &) = default;
    ISerialisable &operator=(const ISerialisable &) = default;
    ISerialisable(ISerialisable &&) = default;
    ISerialisable &operator=(ISerialisable &&) = default;

protected:
    ISerialisable() = default;
};

} // namespace optifol

// ReSharper disable once CppDoxygenUnresolvedReference

/**
 * @class formatter<Serialisable>
 * @brief Helper for C++20 @ref std::format support on serialisable Optifol types.
 * @tparam Serialisable The derived ISerialisable type to format
 */
template<typename Serialisable>
    requires std::derived_from<Serialisable, optifol::ISerialisable>
struct std::formatter<Serialisable> : formatter<string> // NOLINT(*-dcl58-cpp)
{
    /**
     * @brief Format the given sentence according to the implementation-defined serialiser.
     * @param value The Serialisable to serialise with the <code>operator&lt;&lt;</code> call.
     * @param context The streamed formatting context
     * @return The updated formatting context
     */
    auto format(const Serialisable &value, format_context &context) const
    {
        ostringstream output_stream;
        output_stream << value;
```

```
        return formatter<string>::format(output_stream.str(), context);
    }
};

#endif // OPTIFOL_ISERIALISABLE_HPP
```

1.4 Logging.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the statically available log4cxx facilities
 * @author Oliver Dixon
 * @date 2025-03-21
 * @version Development
 */

#include <log4cxx/basicconfigurator.h>
#include <log4cxx/logmanager.h>
#include <log4cxx/propertyconfigurator.h>
#include <log4cxx/xml/domconfigurator.h>

#include "Logging.hpp"

namespace optifol
{

Logging::LifecycleManager Logging::manager;

const char *const Logging::LifecycleManager::properties_file = "Resources/log4cxx.xml";

log4cxx::LoggerPtr Logging::get_logger()
{
    return log4cxx::LogManager::getRootLogger();
}

log4cxx::LoggerPtr Logging::get_logger(const char *name)
{
    return log4cxx::Logger::getLogger(name);
}

log4cxx::LoggerPtr Logging::get_logger(std::vector<std::string> &&name)
{
    const auto component_count = name.size();
    std::string qualifiedName;

    for (std::size_t component_idx = 0; component_idx < component_count - 1; ++component_idx)
        qualifiedName += name[component_idx] + '.';

    return log4cxx::Logger::getLogger(qualifiedName + name[component_count - 1]);
}

Logging::LifecycleManager::LifecycleManager()
{
    if (log4cxx::xml::DOMConfigurator::configure(properties_file) ==
            log4cxx::spi::ConfigurationStatus::NotConfigured) {
        // If we couldn't load the custom configurator, send events to the console.
        log4cxx::BasicConfigurator::configure();
        LOG4CXX_WARN(get_logger(), "Could not load logging properties file at " << properties_file);
    }
}

Logging::LifecycleManager::~LifecycleManager()
{
    log4cxx::LogManager::shutdown();
}

} // namespace optifol
```

1.5 Logging.hpp

```
/*
 * Copyright (c) All Rights Reserved
```

```
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the statically available log4cxx facilities
 * @author Oliver Dixon
 * @date 2025-03-21
 * @version Development
 */

#ifndef LOGGING_HPP
#define LOGGING_HPP

#include <log4cxx/logger.h>

namespace optifol
{

/**
 * @class Logging
 * @brief Provide log4cxx infrastructure from a static context.
 */
class Logging
{
public:
    /**
     * @brief Get the root logger
     * @return A shared pointer to the root log4cxx logger
     */
    static log4cxx::LoggerPtr get_logger();

    /**
     * @brief Get the named logger
     * @param name The period-delimited qualified name of the log4cxx logger to retrieve
     * @return A shared pointer to the named log4cxx logger
     */
    static log4cxx::LoggerPtr get_logger(const char *name);

    /**
     * @brief Get the named logger
     * @param name The field-delimited qualified name of the log4cxx logger to retrieve, with one element per
     *  index
     * @return A shared pointer to the named log4cxx logger
     */
    static log4cxx::LoggerPtr get_logger(std::vector<std::string> &&name);

private:
    static struct LifecycleManager
    {
        LifecycleManager();
        ~LifecycleManager();

    private:
        static const char *const properties_file;
    } manager;
};

} // namespace optifol

#endif
```

1.6 Optifol.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Optifol namespace declaration required for documentary purposes
 * @author Oliver Dixon
 * @date 2025-03-03
 * @version Development
 */

#ifndef OPTIFOL_HPP
#define OPTIFOL_HPP
```

```
// ReSharper disable once CppUnusedIncludeDirective - Referenced by RAPIDJSON_PARSE_ERROR_NORETURN.
#include "Exceptions/ParseError.hpp"

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wmacro-redefined" // Redefinition is intentional.
#define RAPIDJSON_PARSE_ERROR_NORETURN(parse_error_code, offset) throw optifol::ParseError(#parse_error_code)
#pragma clang diagnostic pop

#include <memory>
#include <unordered_map>
#include <unordered_set>

/**
 * @namespace optifol
 * @brief The Optifol namespace contains the vast majority of the symbols defined by the Optifol client
 *  application
 * @note In a proportionally small number of cases, it is necessary to define symbols in the namespaces of the
 *  standard library or various third-party libraries. No such usage is UB, and all such symbol definitions
 *  are the result of template specialisation.
 */
namespace optifol
{

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wdocumentation"
/**
 * @concept WeaklyEqualityComparableWith
 * @brief Modelled if and only if the given types may be tested for equality and inequality using
 *  <code>operator==</code> and <code>operator!=</code> member functions, without requiring a common type.
 * @tparam T The first type
 * @tparam U The second type
 * @details This concept mirrors the standard @ref std::equality_comparable_with, without the requirement to
 * model
 *  @ref std::common_reference_with, hence allowing comparisons without converting constructors.
 */
template<class T, class U>
concept WeaklyEqualityComparableWith = std::equality_comparable<T> && std::equality_comparable<U> &&
        requires(const std::remove_reference_t<T> &t, const std::remove_reference_t<U> &u) {
            { t == u } -> std::convertible_to<bool>;
            { t != u } -> std::convertible_to<bool>;
            { u == t } -> std::convertible_to<bool>;
            { u != t } -> std::convertible_to<bool>;
        };
#pragma clang diagnostic pop

/**
 * @class UniversalTransparentEquality
 * @brief Provides a transparent equality functor for homogenously comparable types through @ref
 * std::unique_ptr,
 *  @ref std::shared_ptr, and raw pointers; or heterogeneously comparing through object references where
 *  WeaklyEqualityComparableWith is modelled.
 * @tparam T The concrete type to transparently compare.
 */
template<typename T>
struct UniversalTransparentEquality
{
    using is_transparent = void;

    // Object reference on LHS

    bool operator()(const T &obj, const T &obj_other) const noexcept
    {
        return obj == obj_other;
    }

    bool operator()(const T &obj, const std::unique_ptr<T> &ptr) const noexcept
    {
        if (ptr == nullptr)
            return false;

        return *ptr == obj;
    }

    bool operator()(const T &obj, const std::shared_ptr<T> &ptr) const noexcept
    {
        if (ptr == nullptr)
            return false;

        return *ptr == obj;
    }
```

```
bool operator()(const T &obj, const T *const ptr) const noexcept
{
    if (ptr == nullptr)
        return false;

    return *ptr == obj;
}

template<typename U>
    requires WeaklyEqualityComparableWith<T, U>
bool operator()(const T &obj, U &&other) const noexcept
{
    return obj == other;
}

// Unique pointer on LHS

bool operator()(const std::unique_ptr<T> &ptr, const std::unique_ptr<T> &ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
        return ptr == ptr_other;

    return *ptr == *ptr_other;
}

bool operator()(const std::unique_ptr<T> &ptr, const T &obj) const noexcept
{
    if (ptr == nullptr)
        return false;

    return *ptr == obj;
}

bool operator()(const std::unique_ptr<T> &ptr, const std::shared_ptr<T> &ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
        return ptr == ptr_other;

    return *ptr == *ptr_other;
}

bool operator()(const std::unique_ptr<T> &ptr, const T *const ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
        return ptr == ptr_other;

    return *ptr == *ptr_other;
}

template<typename U>
    requires WeaklyEqualityComparableWith<T, U>
bool operator()(const std::unique_ptr<T> &ptr, U &&other) const noexcept
{
    if (ptr == nullptr)
        return false;

    return *ptr == other;
}

// Shared pointer on LHS

bool operator()(const std::shared_ptr<T> &ptr, const std::shared_ptr<T> &ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
        return ptr == ptr_other;

    return *ptr == *ptr_other;
}

bool operator()(const std::shared_ptr<T> &ptr, const T &obj) const noexcept
{
    if (ptr == nullptr)
        return false;

    return *ptr == obj;
}

bool operator()(const std::shared_ptr<T> &ptr, const std::unique_ptr<T> &ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
```

```
        return ptr == ptr_other;

    return *ptr == *ptr_other;
}

bool operator()(const std::shared_ptr<T> &ptr, const T *const ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
        return ptr == ptr_other;

    return *ptr == *ptr_other;
}

template<typename U>
    requires WeaklyEqualityComparableWith<T, U>
bool operator()(const std::shared_ptr<T> &ptr, U &&other) const noexcept
{
    if (ptr == nullptr)
        return false;

    return *ptr == other;
}

// Raw pointer on LHS

bool operator()(const T *const ptr, const T *const ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
        return ptr == ptr_other;

    return ptr->operator==(*ptr_other);
}

bool operator()(const T *const ptr, const T &obj) const noexcept
{
    if (ptr == nullptr)
        return false;

    return *ptr == obj;
}

bool operator()(const T *const ptr, const std::unique_ptr<T> &ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
        return ptr == ptr_other;

    return *ptr == *ptr_other;
}

bool operator()(const T *const ptr, const std::shared_ptr<T> &ptr_other) const noexcept
{
    if (ptr == nullptr || ptr_other == nullptr)
        return ptr == ptr_other;

    return *ptr == *ptr_other;
}

template<typename U>
    requires WeaklyEqualityComparableWith<T, U>
bool operator()(const T *const ptr, U &&other) const noexcept
{
    if (ptr == nullptr)
        return false;

    return *ptr == other;
}

// Dynamically typed object on LHS

template<typename U>
    requires WeaklyEqualityComparableWith<T, U>
bool operator()(U &&obj, const T &obj_other) const noexcept
{
    return obj == obj_other;
}

template<typename U>
    requires WeaklyEqualityComparableWith<T, U>
bool operator()(U &&obj, const std::unique_ptr<T> &ptr_other) const noexcept
{
    if (ptr_other == nullptr)
```

```
            return false;

        return obj == *ptr_other;
    }

    template<typename U>
        requires WeaklyEqualityComparableWith<T, U>
    bool operator()(U &&obj, const std::shared_ptr<T> &ptr_other) const noexcept
    {
        if (ptr_other == nullptr)
            return false;

        return obj == *ptr_other;
    }

    template<typename U>
        requires WeaklyEqualityComparableWith<T, U>
    bool operator()(U &&obj, const T *const ptr_other) const noexcept
    {
        if (ptr_other == nullptr)
            return false;

        return obj == *ptr_other;
    }

    template<typename U>
        requires WeaklyEqualityComparableWith<T, U>
    bool operator()(U &&obj, U &&other_obj) const noexcept = delete;
};


/**
 * @typedef UniqueUnorderedSet
 * @brief The @ref std::unordered_set collection to store the templated type <code>T</code> within a
 *  transparently hashable and comparable @ref std::unique_ptr.
 * @tparam T The type to store, detained within @ref std::unique_ptr.
 */
template<typename T>
using UniqueUnorderedSet =
        std::unordered_set<std::unique_ptr<T>, std::hash<T>, UniversalTransparentEquality<T>>;

/**
 * @typedef UniqueUnorderedMap
 * @brief The @ref std::unordered_map collection to store the templated type <code>T</code> within a
 *  transparently hashable and comparable @ref std::unique_ptr.
 * @tparam K The key type to store, detained within @ref std::unique_ptr.
 * @tparam V The value type to associate with keys.
 */
template<typename K, typename V>
using UniqueUnorderedMap =
        std::unordered_map<std::unique_ptr<K>, V, std::hash<K>, UniversalTransparentEquality<K>>;

/**
 * @typedef SharedUnorderedSet
 * @brief The @ref std::unordered_set collection to store the templated type <code>T</code> within a
 *  transparently hashable and comparable @ref std::shared_ptr.
 * @tparam T The type to store, detained within @ref std::shared_ptr.
 */
template<typename T>
using SharedUnorderedSet =
        std::unordered_set<std::shared_ptr<T>, std::hash<T>, UniversalTransparentEquality<T>>;

/**
 * @typedef SharedUnorderedMap
 * @brief The @ref std::unordered_map collection to store the templated type <code>T</code> within a
 *  transparently hashable and comparable @ref std::shared_ptr.
 * @tparam K The key type to store, detained within @ref std::shared_ptr.
 * @tparam V The value type to associate with keys.
 */
template<typename K, typename V>
using SharedUnorderedMap =
        std::unordered_map<std::shared_ptr<K>, V, std::hash<K>, UniversalTransparentEquality<K>>;

/**
 * @typedef RawUnorderedSet
 * @brief The @ref std::unordered_set collection to store the templated type <code>T</code> within a
 *  transparently hashable and comparable <code>T*</code>.
 * @tparam T The type to store, referenced through a raw pointer.
 */
template<typename T>
using RawUnorderedSet =
```

```
        std::unordered_set<std::add_pointer_t<T>, std::hash<T>, UniversalTransparentEquality<T>>;

/**
 * @typedef RawUnorderedMap
 * @brief The @ref std::unordered_map collection to store the templated type <code>T</code> within a
 *  transparently hashable and comparable <code>T*</code>.
 * @tparam K The key type to store, referenced through a raw pointer.
 * @tparam V The value type to associate with keys.
 */
template<typename K, typename V>
using RawUnorderedMap =
        std::unordered_map<std::add_pointer_t<K>, V, std::hash<K>, UniversalTransparentEquality<K>>;

} // namespace optifol

#endif // OPTIFOL_HPP
```

1.7 Exceptions

1.7.1 ParseError.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification and implementation for the Parse Error exception
 * @date 2025-05-09
 * @author Oliver Dixon <od641@york.ac.uk>
 */

#ifndef PARSEERROR_HPP
#define PARSEERROR_HPP

#include <stdexcept>

namespace optifol
{

/**
 * @class ParseError
 * @brief An exception indicating some sort of parsing error
 */
class ParseError final : public std::runtime_error
{
public:
    explicit ParseError(const std::string &message) :
        std::runtime_error(std::string("Parsing Error: ") + message)
    {
    }

    explicit ParseError(const std::string &message, const std::size_t) :
        std::runtime_error(std::string("Parsing Error: ") + message)
    {
    }

    explicit ParseError(const char *message, const std::size_t) :
        std::runtime_error(std::string("Parsing Error: ") + message)
    {
    }
};

} // namespace optifol

#endif
```

1.7.2 SemanticException.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
```

```
 * @brief Class specification and implementation for the Semantic Exception
 * @author Oliver Dixon
 * @date 2024-11-30
 * @version Development
 */

#ifndef SEMANTICEXCEPTION_HPP
#define SEMANTICEXCEPTION_HPP

#include <stdexcept>

namespace optifol
{

/**
 * @class SemanticException
 * @brief The Semantic Exception expresses a logical/semantic error in the meaning of a Requirement statement.
 * @details The Semantic Exception can be used to express errors of a nature that, while syntactically
 *  unproblematic, do not make meaningful sense in the present context. The most canonical usage would be to
 *  indicate a logical inconsistency in a requirement sentence according to the rules of FOL. See, for
 *  example, SymbolStandardisingVisitor.
 */
class SemanticException final : public std::runtime_error
{
public:
    explicit SemanticException(const std::string &message) :
        std::runtime_error(std::string("Semantic Exception: ") + message)
    {
    }

    explicit SemanticException(const char *message) :
        std::runtime_error(std::string("Semantic Exception: ") + message)
    {
    }
};

} // namespace optifol

#endif
```

1.8 GUI

1.8.1 Application.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Optifol GTK Application
 * @author Oliver Dixon
 * @date 2025-02-02
 * @version Development
 */

#include <iostream>

#include "../Storage/Subsystem.hpp"
#include "Application.hpp"
#include "GTKHelpers.hpp"
#include "MainWindow.hpp"

namespace optifol
{

Glib::RefPtr<Application> Application::create()
{
    return Glib::make_refptr_for_instance<Application>(new Application());
}

Application::Application() :
    Gtk::Application("uk.ac.york.www_users.od641.optifol")
{
    Glib::set_application_name("Optifol");
}

void Application::on_startup()
```

```
{
    Gtk::Application::on_startup();

    add_action("new", [] { std::cout << "New\n"; });
    add_action("open", [] { std::cout << "Open\n"; });
    add_action("quit", sigc::mem_fun(*this, &Application::quit_application));
    add_action("about", sigc::mem_fun(*this, &Application::show_about_dialog));
    add_action("help", &Application::open_application_help);

    builder = Gtk::Builder::create_from_resource("/uk/ac/york/www_users/od641/optifol/UI/Application.ui");
    set_menubar(GTKHelpers::get_object<Gio::Menu>("Application Root", *builder, "top_menu"));
    about_dialog = GTKHelpers::get_widget<Gtk::AboutDialog>("Application Root", *builder, "about_dialog");
    about_dialog->set_hide_on_close();
}

void Application::on_activate()
{
    const auto main_window = create_main_window();
    main_window->present();
    about_dialog->set_transient_for(*main_window);

    // ReSharper disable once CppDFAMemoryLeak
}

MainWindow *Application::create_main_window()
{
    // Memory leak warning here is a false positive, as windows are managed by the GTK management engine
    // ReSharper disable once CppDFAMemoryLeak
    const auto main_window = new MainWindow();
    add_window(*main_window);

    main_window->set_show_menubar();
    main_window->signal_hide().connect([main_window]() { delete main_window; });

    return main_window;
}

void Application::show_about_dialog() const
{
    about_dialog->set_visible();
    about_dialog->present();
}

void Application::open_application_help()
{
    Gio::AppInfo::launch_default_for_uri("https://www-users.york.ac.uk/~od641/l6-project");
}

void Application::quit_application()
{
    quit();

    // Destruct all windows constituting the Application instance.
    for (const auto windows = get_windows(); const auto window: windows)
        window->set_visible(false);
}

} // namespace optifol
```

1.8.2 Application.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Optifol GTK Application
 * @author Oliver Dixon
 * @date 2025-02-02
 * @version Development
 */

#ifndef APPLICATION_HPP
#define APPLICATION_HPP

#include <gtkmm.h>
```

```
namespace optifol
{

class MainWindow;

/**
 * @class Application
 * @brief The Optifol application managing startup, shutdown, the main window, and core user actions.
 */
class Application : public Gtk::Application
{
public:
    /**
     * @brief Create a new Application from a static context, producing a reference to the created Application
     * in the GTK memory model.
     * @return
     */
    static Glib::RefPtr<Application> create();

protected:
    /**
     * @brief Create the Application from a non-static context.
     */
    Application();

    void on_startup() override;

    void on_activate() override;

private:
    /**
     * @brief Create the MainWindow and return a mutable, observing pointer.
     * @return A weak non-owning reference to the created MainWindow.
     */
    MainWindow *create_main_window();

    /**
     * @brief Display the 'About' dialog.
     */
    void show_about_dialog() const;

    /**
     * @brief Display the 'Help' web page in the default system browser.
     */
    static void open_application_help();

    /**
     * @brief Quit the application.
     */
    void quit_application();

    Glib::RefPtr<Gtk::Builder> builder;

    Gtk::AboutDialog *about_dialog = nullptr;
};

} // namespace optifol

#endif
```

1.8.3 ContextButtonCorrespondence.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Context Menu-Menu Button correspondence helper
 * @author Oliver Dixon
 * @date 2026-04-11
 * @version Development
 */

#include <gtkmm/gestureclick.h>

#include "ContextButtonCorrespondence.hpp"
```

```
namespace optifol
{

ContextButtonCorrespondence::ContextButtonCorrespondence(Gtk::Widget *parent,
        const Glib::RefPtr<Gio::Menu> &menu_source,
        std::initializer_list<std::tuple<std::string, Gtk::MenuButton *, Gtk::Popover *, bool>> &&map)
{
    /*
     * Establish the mapping between popover-raising actions and corresponding MenuButtons. If a popover
     * constructor has been defined, call that immediately prior to the popup call.
     */
    for (const auto &[name, menu_button, popover, enabled]: map) {
        const auto action_source = action_group->add_action(name, [popover] { popover->popup(); });

        this->map.emplace(name, menu_button);
        menu_button->set_popover(*popover);
        menu_button->set_sensitive(enabled);
        action_source->set_enabled(enabled);
    }

    /*
     * Set up a callback for the changing state of actions. When the state of an underlying Gio::SimpleAction
     * is modified, this callback is triggered to locate the corresponding Gtk::MenuButton and toggle its
     * sensitivity accordingly. Therefore, the graphical button state always matches the availability of the
     * action.
     */
    action_group->signal_action_enabled_changed().connect(
            [this](const Glib::ustring &action_name, const bool enabled)
            {
                const auto map_it = this->map.find(action_name);
                if (map_it != this->map.cend())
                    map_it->second->set_sensitive(enabled);
            });

    // Set up the PopoverMenu context menu, including registration with the parent Widget.
    menu.set_parent(*parent);
    menu.set_menu_model(menu_source);

    const auto gesture = Gtk::GestureClick::create();
    gesture->set_button(GDK_BUTTON_SECONDARY);
    gesture->signal_released().connect(
            [this](int, const double x, const double y)
            {
                const Gdk::Rectangle rect(static_cast<int>(x), static_cast<int>(y), 1, 1);
                menu.set_pointing_to(rect);
                menu.popup();
            });

    parent->insert_action_group("win", action_group);
    parent->add_controller(gesture);
}

void ContextButtonCorrespondence::enable_action(const Glib::ustring &action_name) const
{
    const auto action =
            std::dynamic_pointer_cast<Gio::SimpleAction>(action_group->lookup_action(action_name));
    if (action != nullptr)
        action->set_enabled(true);
}

void ContextButtonCorrespondence::disable_action(const Glib::ustring &action_name) const
{
    const auto action =
            std::dynamic_pointer_cast<Gio::SimpleAction>(action_group->lookup_action(action_name));
    if (action != nullptr)
        action->set_enabled(false);
}

} // namespace optifol
```

1.8.4 ContextButtonCorrespondence.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
```

```
 * @brief Class specification for the Context Menu–Menu Button correspondence helper
 * @author Oliver Dixon
 * @date 2026-04-11
 * @version Development
 */

#ifndef CONTEXTBUTTONCORRESPONDENCE_HPP
#define CONTEXTBUTTONCORRESPONDENCE_HPP

#include <giomm/menu.h>
#include <giomm/simpleactiongroup.h>
#include <gtkmm/menubutton.h>
#include <gtkmm/popovermenu.h>

namespace optifol
{

/**
 * @class ContextButtonCorrespondence
 * @brief Manage common popover-based actions linked by both a context menu item and menu button.
 */
class ContextButtonCorrespondence
{
public:
    /**
     * @brief Create a new correspondence
     * @param parent The parent widget of the PopoverMenu containing the menu option
     * @param menu_source The GTK list model populating the PopoverMenu
     * @param map A tuple containing the action name, the menu button to which it is linked, the popover
     * linked to the action, and whether it should be enabled by default.
     */
    explicit ContextButtonCorrespondence(Gtk::Widget *parent, const Glib::RefPtr<Gio::Menu> &menu_source,
            std::initializer_list<std::tuple<std::string, Gtk::MenuButton *, Gtk::Popover *, bool>> &&map);

    /**
     * @brief Enable the named action via its menu button and context menu entry.
     * @param action_name The name of the action to enable
     */
    void enable_action(const Glib::ustring &action_name) const;

    /**
     * @brief Disable the named action via its menu button and context menu entry.
     * @param action_name The name of the action to disable
     */
    void disable_action(const Glib::ustring &action_name) const;

private:
    std::unordered_map<std::string, Gtk::MenuButton *> map;

    Glib::RefPtr<Gio::SimpleActionGroup> action_group = Gio::SimpleActionGroup::create();

    Gtk::PopoverMenu menu;
};

} // namespace optifol

#endif
```

1.8.5 GTKHelpers.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for some handy exception-aware GTK static helpers
 * @author Oliver Dixon
 * @date 2025-02-02
 * @version Development
 */

#ifndef GTKHELPERS_HPP
#define GTKHELPERS_HPP

#include <assert.h>
#include <gtkmm.h>
```

```cpp
#include "../Storage/StorageObjectBase.hpp"

namespace optifol
{

/**
 * @namespace mp_helpers
 * @brief Metaprogramming UI helpers that are required to be declared at the namespace level
 */
namespace mp_helpers
{

/**
 * @class is_optional
 * @brief False case for testing specialisations of @ref std::optional
 */
template<typename>
struct is_optional : std::false_type
{
};

/**
 * @class is_optional
 * @brief True case for testing specialisations of @ref std::optional
 * @tparam T The type to test
 */
template<typename T>
struct is_optional<std::optional<T>> : std::true_type
{
};

// Current Clang 18 bug reports Doxygen violations for uses of @tparam on templated concepts.
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wdocumentation"

/**
 * @concept GTKWidget
 * @brief Represents a concrete GTK widget
 * @tparam WidgetType The concrete widget type
 */
template<typename WidgetType>
concept GTKWidget = std::derived_from<WidgetType, Gtk::Widget> && !std::is_abstract_v<WidgetType>;

/**
 * @concept GTKObject
 * @brief Represents a concrete GTK object
 * @tparam ObjectType The concrete object type
 */
template<typename ObjectType>
concept GTKObject = std::derived_from<ObjectType, Glib::Object> && !std::is_abstract_v<ObjectType>;

#pragma clang diagnostic pop

} // namespace mp_helpers

/**
 * @class GTKHelpers
 * @brief A handy set of modern exception-aware (throwing) wrappers for common GTKmm operations
 */
class GTKHelpers
{
public:
    /**
     * @brief Retrieve a GTK object as a Glib-wrapped ref-counted pointer from a GTK builder
     * @tparam ObjectType The expected concrete GTK object return type from the builder
     * @param segment_name A view of the name of the context requesting the item; used for enhanced error
     *  logging.
     * @param builder The builder with an expected reference to the item
     * @param object_name The ID of the target GTK object
     * @return A Glib-wrapped instance of the GTK object typed according to the template parameter
     * @throws std::runtime_error The named object could not be built according to the given type and name
     */
    template<mp_helpers::GTKObject ObjectType>
    static Glib::RefPtr<ObjectType> get_object(
            const std::string_view segment_name, Gtk::Builder &builder, const std::string &object_name)
    {
        const auto object = builder.get_object<ObjectType>(object_name);
        if (!object)
            throw std::runtime_error("Could not build \"" + std::string(segment_name) + "\": GTK object \"" +
                    object_name + "\" was not found");
        return object;
```

```
}

/**
 * @brief Retrieve a GTK widget as a raw pointer (ultimately managed by the GTK object system) from a GTK
 *  builder
 * @tparam WidgetType The expected concrete GTK widget return type from the builder
 * @param segment_name A view of the name of the context requesting the item; used for enhanced error
 *  logging.
 * @param builder The builder with an expected reference to the item
 * @param widget_name The ID of the target GTK widget
 * @return A Glib-wrapped instance of the GTK widget typed according to the template parameter
 * @throws std::runtime_error The named widget could not be built according to the given type and name
 */
template<mp_helpers::GTKWidget WidgetType>
static WidgetType *get_widget(
        const std::string_view segment_name, Gtk::Builder &builder, const std::string &widget_name)
{
    const auto widget = builder.get_widget<WidgetType>(widget_name);
    if (!widget)
        throw std::runtime_error("Could not build \"" + std::string(segment_name) + "\": GTK widget \"" +
                widget_name + "\" was not found");

    return widget;
}

/**
 * @brief Set up a non-expandable GTK label within the given container
 * @param list_item The container into which the label should be emplaced
 * @param mono_styling Should the label be styled according to the standard monospace style?
 */
static void setup_label(const Glib::RefPtr<Gtk::ListItem> &list_item, const bool mono_styling = false)
{
    const auto label = Gtk::make_managed<Gtk::Label>();

    label->set_halign(Gtk::Align::START);
    if (mono_styling)
        label->add_css_class("optifol_monospace");

    list_item->set_child(*label);
}

/**
 * @brief Set up an editable text field (Gtk::Entry) within the given container
 * @param list_item The container into which the field should be emplaced
 */
static void setup_entry(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto entry = Gtk::make_managed<Gtk::Entry>();
    entry->set_halign(Gtk::Align::START);
    list_item->set_child(*entry);
}

/**
 * @brief Set up a combo box (Gtk::DropDown) within the given container
 * @param list_item The container into which the combo box should be emplaced
 */
static void setup_combo_box(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto combo_box = Gtk::make_managed<Gtk::DropDown>();
    combo_box->set_halign(Gtk::Align::START);
    list_item->set_child(*combo_box);
}

/**
 * @brief Set up an expandable GTK label within the given container
 * @param list_item The container into which the label should be emplaced
 * @param mono_styling Should the label be styled according to the standard monospace style?
 */
static void setup_expandable_label(
        const Glib::RefPtr<Gtk::ListItem> &list_item, const bool mono_styling = false)
{
    const auto expander = Gtk::make_managed<Gtk::TreeExpander>();
    const auto label = Gtk::make_managed<Gtk::Label>();

    label->set_halign(Gtk::Align::START);
    if (mono_styling)
        label->add_css_class("optifol_monospace");

    expander->set_child(*label);
    list_item->set_child(*expander);
}
```

```
    /**
     * @brief Bind an arbitrary optional Glib::Property to a Gtk::Label
     * @tparam BoundType The @ref std::optional specialisation type to be stringified and bound
     * @tparam StoredType The class providing the property getter
     * @param property_functor Unbound member function functor to provide observing property proxy, i.e.
     *  Glib::PropertyProxy_ReadOnly.
     * @param object The object instance containing the property to be bound
     * @param label The destination label to contain a string representation of the property
     * @details If the property functor provides a @ref std::optional containing a value, the populated label
     *  is equivalent to the one provided by the non-@ref std::optional @ref bind_any_property. If the
     *  supplied property does not contain a value, the label is marked "Empty" and styled with the
     *  @ref unknown_css_class_name CSS class.
     */
    template<typename BoundType, typename StoredType>
        requires mp_helpers::is_optional<BoundType>::value
    static void bind_any_property(
            sigc::mem_functor<Glib::PropertyProxy_ReadOnly<BoundType> (StoredType::*)() const>
                    property_functor,
            const StoredType &object, Gtk::Label *const label)
    {
        if (label == nullptr)
            return;

        Glib::Binding::bind_property(property_functor.operator()(object), label->property_label(),
                Glib::Binding::Flags::SYNC_CREATE,
                [label](const BoundType &from) -> std::optional<Glib::ustring>
                {
                    const bool is_already_unknown = label->has_css_class(unknown_css_class_name);
                    bool unknown_value = false;
                    std::string string_value;

                    if (from.has_value()) {
                        if constexpr (std::is_convertible_v<Glib::ustring, decltype(*from)>)
                            // If we have plain string value, just pass it through.
                            string_value = *from;
                        else
                            // Otherwise, rely on the standard conversion functions with ADR.
                            string_value = std::to_string(*from);
                    } else {
                        unknown_value = true;
                        string_value = "Empty";
                    }

                    if (is_already_unknown && !unknown_value)
                        label->remove_css_class(unknown_css_class_name);
                    else if (!is_already_unknown && unknown_value)
                        label->add_css_class(unknown_css_class_name);

                    return string_value;
                });
    }

    /**
     * @brief Bind an arbitrary non-optional Glib::Property to a Gtk::Label
     * @tparam BoundType The type to be stringified and bound
     * @tparam StoredType The class providing the property getter
     * @param property_functor Unbound member function functor to provide observing property proxy, i.e.
     *  Glib::PropertyProxy_ReadOnly.
     * @param object The object instance containing the property to be bound
     * @param label The destination label to contain a string representation of the property
     */
    template<typename BoundType, typename StoredType>
    static void bind_any_property(
            sigc::mem_functor<Glib::PropertyProxy_ReadOnly<BoundType> (StoredType::*)() const>
                    property_functor,
            const StoredType &object, Gtk::Label *const label)
    {
        if (label == nullptr)
            return;

        Glib::Binding::bind_property(property_functor.operator()(object), label->property_label(),
                Glib::Binding::Flags::SYNC_CREATE);
    }

private:
    static constexpr auto unknown_css_class_name = "optifol_unknown";
};

} // namespace optifol
```

```
#endif
```

1.8.6 GUIDriver.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Optifol GUI entry point definition
 * @author Oliver Dixon
 * @date 2025-02-02
 * @version Development
 */

#include <log4cxx/basicconfigurator.h>

#include "Application.hpp"

int main(const int argc, char **argv)
{
    log4cxx::BasicConfigurator::configure();
    const auto app = optifol::Application::create();
    return app->run(argc, argv);
}
```

1.8.7 IWindowArea.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Interface specification for the Subsystem-sensitive Window Area
 * @author Oliver Dixon
 * @date 2025-07-09
 * @version Development
 */

#ifndef IWINDOWAREA_HPP
#define IWINDOWAREA_HPP

#include <glibmm/refptr.h>
#include <sigc++/trackable.h>

namespace optifol
{

class Subsystem;

/**
 * @class IWindowArea
 * @brief A window area is a loosely defined "major" area of the Optifol GUI that is largely dependent on the
 *  presently selected model in the ProjectHierarchyPane. It is trackable to support the usage of its member
 *  functions as libsigc++ callbacks.
 */
class IWindowArea : public sigc::trackable
{
public:
    /**
     * @brief Create a new window area
     */
    IWindowArea() = default;

    // ReSharper disable once CppHidingFunction
    /*
     * A well-document and understood sigc++ constraint prevents deletions through sigc::trackable pointers.
     * Hiding the non-virtual ~sigc::trackable is a non-issue for our API use-case.
     */
    /**
     * @brief Destruct the window area
     */
    virtual ~IWindowArea() = default;
```

```
    /**
     * @brief Disallow copying of entire areas, as semantically invalid and presumed to be singleton.
     */
    IWindowArea(const IWindowArea &) = delete;

    /**
     * @brief Disallow moving of entire areas, as semantically invalid and presumed to be singleton.
     */
    IWindowArea(IWindowArea &&) = delete;

    /**
     * @brief Handle a change in the present selection to a new Subsystem model
     * @param subsystem_model The newly selected Subsystem model
     * @see ProjectHierarchyPane::add_subsystem_change_callback
     */
    virtual void select_model(const Glib::RefPtr<Subsystem> &subsystem_model) = 0;

    /**
     * @brief Handle a deselection (and no re-selection) of the previously selected Subsystem model
     * @see ProjectHierarchyPane::add_subsystem_change_callback
     */
    virtual void deselect_model() = 0;

    /**
     * @brief Retrieves a mutating raw pointer to the active Subsystem loaded into the area
     * @return The mutating pointer, designed to be use for transitory single-threaded use only due to lack of
     *  documented lifetime guarantees.
     */
    virtual Subsystem *get_active_subsystem() noexcept = 0;

    /**
     * @brief Retrieves an observing raw pointer to the active Subsystem loaded into the area
     * @return The observing pointer, designed to be use for transitory single-threaded use only due to lack
     *  of documented lifetime guarantees.
     */
    virtual const Subsystem *observe_active_subsystem() const noexcept = 0;
};

} // namespace optifol

#endif // IWINDOWAREA_HPP
```

1.8.8 MainWindow.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Optifol GTK main window
 * @author Oliver Dixon
 * @date 2025-02-02
 * @version Development
 */

#include "MainWindow.hpp"

#include "GTKHelpers.hpp"
#include "TestingArea/TestingArea.hpp"

namespace optifol
{

const char *const MainWindow::area_name = "Main Window";

MainWindow::MainWindow() :
    builder(Gtk::Builder::create_from_resource("/uk/ac/york/www_users/od641/optifol/UI/MainWindow.ui")),
    root_grid(GTKHelpers::get_widget<Gtk::Box>(area_name, *builder, "root_grid"))
{
    set_title("Optifol");
    set_child(*root_grid);

    requirements_index_area = std::make_unique<RequirementsIndexArea>(*builder);
    analysis_area = std::make_unique<AnalysisArea>(*builder);
    testing_area = std::make_unique<TestingArea>(*builder);
    reports_area = std::make_unique<ReportsArea>(*builder);
```

```
    project_hierarchy_pane =
            std::make_unique<ProjectHierarchyPane>(*builder, Gio::ListStore<Project>::create());

    // Register the Requirements Index Area as a Subsystem-sensitive area.
    project_hierarchy_pane->add_subsystem_change_callback(
            sigc::mem_fun(*requirements_index_area, &RequirementsIndexArea::select_model),
            sigc::mem_fun(*requirements_index_area, &RequirementsIndexArea::deselect_model));

    // Register the Analysis Area as a Subsystem-sensitive area.
    project_hierarchy_pane->add_subsystem_change_callback(
            sigc::mem_fun(*analysis_area, &AnalysisArea::select_model),
            sigc::mem_fun(*analysis_area, &AnalysisArea::deselect_model));

    // Register the Testing Area as a Subsystem-sensitive area.
    project_hierarchy_pane->add_subsystem_change_callback(
            sigc::mem_fun(*testing_area, &TestingArea::select_model),
            sigc::mem_fun(*testing_area, &TestingArea::deselect_model));

    // Register the Reports Area as a Subsystem-sensitive area.
    project_hierarchy_pane->add_subsystem_change_callback(
            sigc::mem_fun(*reports_area, &ReportsArea::select_model),
            sigc::mem_fun(*reports_area, &ReportsArea::deselect_model));

    const auto css_provider = Gtk::CssProvider::create();
    Gtk::StyleProvider::add_provider_for_display(
            get_display(), css_provider, GTK_STYLE_PROVIDER_PRIORITY_APPLICATION);
    css_provider->load_from_resource("/uk/ac/york/www_users/od641/optifol/UI/MainWindow/styles.css");
}

} // namespace optifol
```

1.8.9 MainWindow.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Optifol GTK main window
 * @author Oliver Dixon
 * @date 2025-02-02
 * @version Development
 */

#ifndef MAINWINDOW_HPP
#define MAINWINDOW_HPP

#include <gtkmm/applicationwindow.h>
#include <gtkmm/box.h>
#include <gtkmm/builder.h>

#include "AnalysisArea/AnalysisArea.hpp"
#include "ProjectHierarchyPane/ProjectHierarchyPane.hpp"
#include "ReportsArea/ReportsArea.hpp"
#include "RequirementsIndexArea/RequirementsIndexArea.hpp"
#include "TestingArea/TestingArea.hpp"

namespace optifol
{

/**
 * @class MainWindow
 * @brief GTK class representing the main Optifol window
 */
class MainWindow : public Gtk::ApplicationWindow
{
public:
    /**
     * @brief Construct a new main window
     */
    MainWindow();

private:
    Glib::RefPtr<Gtk::Builder> builder;
    static const char *const area_name;
```

```
    Gtk::Box *const root_grid;

    std::unique_ptr<ProjectHierarchyPane> project_hierarchy_pane;

    std::unique_ptr<RequirementsIndexArea> requirements_index_area;
    std::unique_ptr<AnalysisArea> analysis_area;
    std::unique_ptr<TestingArea> testing_area;
    std::unique_ptr<ReportsArea> reports_area;
};

} // namespace optifol

#endif
```

1.8.10 ProcessExecutor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the external Process Executor
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#include <glibmm/main.h>

#include "../Logging.hpp"
#include "ProcessExecutor.hpp"

namespace optifol
{

const log4cxx::LoggerPtr ProcessExecutor::logger =
        Logging::get_logger({"SubprocessControl", "ProcessExecutor"});

ProcessExecutor::ProcessExecutor(const std::string &working_directory, const std::vector<std::string> &argv,
        const std::vector<std::string> &envp, sigc::slot<void(int)> &&finished_callback) :
    finished_callback(std::move(finished_callback))
{
    try {
        Glib::spawn_async_with_pipes(working_directory, argv, envp,
                Glib::SpawnFlags::SEARCH_PATH | Glib::SpawnFlags::DO_NOT_REAP_CHILD, {}, &pid, nullptr,
                &stdout_fd, &stderr_fd);
    } catch (const Glib::SpawnError &spawn_error) {
        logger->error("Failed to spawn sub-process.");
        logger->error(spawn_error.what());
        throw;
    }

    Glib::signal_child_watch().connect(sigc::mem_fun(*this, &ProcessExecutor::reap_child), pid);
    logger->info("Spawned sub-process \"" + argv[0] + "\" with PID " + std::to_string(pid) + '.');
}

void ProcessExecutor::reap_child(const Glib::Pid ended_pid, const int exit_code) noexcept
{
    if (ended_pid != pid)
        // Filter PIDs that aren't ours. (Shouldn't ever happen, but isn't worth logging.)
        return;

    Glib::spawn_close_pid(pid);

    if (exit_code == 0)
        logger->info("Subprocess with PID " + std::to_string(pid) + " exited normally.");
    else
        logger->warn("Subprocess with PID " + std::to_string(pid) + " exited with non-zero exit code " +
                std::to_string(exit_code) + '.');

    finished_callback(exit_code);
}

} // namespace optifol
```

1.8.11 ProcessExecutor.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the external non-streaming Process Executor
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#ifndef PROCESSEXECUTOR_HPP
#define PROCESSEXECUTOR_HPP

#include <glibmm/spawn.h>
#include <log4cxx/logger.h>
#include <string>
#include <vector>

namespace optifol
{

/**
 * @class ProcessExecutor
 * @brief Provides a RAII container for asynchronous execution of external processes under the Glib framework.
 * @details The ProcessExecutor wraps the @ref Glib::spawn_async_with_pipes in an RAII container and provides
 *  a callback option to notify users of subprocess termination.
 */
class ProcessExecutor : public sigc::trackable
{
public:
    /**
     * @brief Asynchronously execute an external process at the given working directory, with the given
     *  argument vector, optionally including a key-value environment specification and exit callback.
     * @param working_directory The working directory in which the sub-process should be started.
     * @param argv The argument vector, where the first entry is the command to execute. The system PATH will
     *  be searched to find the executable, but it is not run under a shell.
     * @param envp The optional <code>KEY=VALUE</code> set of environment variables to export into the
     *  sub-process environment.
     * @param finished_callback The noexcept callback to execute on completion of the sub-process, taking the
     *  numerical exit code.
     */
    ProcessExecutor(const std::string &working_directory, const std::vector<std::string> &argv,
            const std::vector<std::string> &envp = {}, sigc::slot<void(int)> &&finished_callback = {});

    // ReSharper disable once CppHidingFunction
    /*
     * A well-document and understood sigc++ constraint prevents deletions through sigc::trackable pointers.
     * Hiding the non-virtual ~sigc::trackable is a non-issue for our API use-case.
     */
    /**
     * @brief Destruct the ProcessExecutor, disconnecting any signals.
     */
    virtual ~ProcessExecutor() = default;

protected:
    /**
     * @brief Reap the child process identified with the given PID, and invoke the user-supplied @ref
     *  finished_callback.
     * @param ended_pid The PID of the ended process.
     * @param exit_code The numerical exit code of the process; anything except zero is deemed indicative of
     *  an error.
     */
    virtual void reap_child(Glib::Pid ended_pid, int exit_code) noexcept;

    int stdout_fd = -1;
    int stderr_fd = -1;
    Glib::Pid pid = -1;

private:
    static const log4cxx::LoggerPtr logger;

    const sigc::slot<void(int)> finished_callback;
};

} // namespace optifol
```

```
#endif // PROCESSEXECUTOR_HPP
```

1.8.12 StreamingProcessExecutor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the external streaming Process Executor
 * @author Oliver Dixon
 * @date 2025-07-27
 * @version Development
 */

#include <cassert>
#include <glibmm/main.h>

#include "../Logging.hpp"
#include "StreamingProcessExecutor.hpp"

namespace optifol
{

const log4cxx::LoggerPtr StreamingProcessExecutor::logger =
        Logging::get_logger({"SubprocessControl", "ProcessExecutor", "PipeStreamers"});

StreamingProcessExecutor::StreamingProcessExecutor(const std::string &working_directory,
        const std::vector<std::string> &argv, const std::vector<std::string> &envp,
        const Glib::RefPtr<Gtk::TextBuffer> &output, sigc::slot<void(int)> &&finished_callback) :
    ProcessExecutor(working_directory, argv, envp, std::move(finished_callback)),
    output(output)
{
    stdout_stream.emplace(get_colour_tag(output, "stdout_tag", "black"), stdout_fd,
            sigc::mem_fun(*this, &StreamingProcessExecutor::stream_callback));

    stderr_stream.emplace(get_colour_tag(output, "stderr_tag", "red"), stderr_fd,
            sigc::mem_fun(*this, &StreamingProcessExecutor::stream_callback));

    logger->debug("For PID " + std::to_string(pid) + ", tracking stdout on FD " + std::to_string(stdout_fd) +
            " and stderr on FD " + std::to_string(stderr_fd) + '.');
}

void StreamingProcessExecutor::write_exception_error(
        const Glib::SpawnError &exception, const Glib::RefPtr<Gtk::TextBuffer> &output)
{
    output->insert_with_tag(output->end(), exception.what(), get_colour_tag(output, "stderr_tag", "red"));
}

bool StreamingProcessExecutor::stream_callback(
        const Glib::IOCondition condition, const Stream *const stream_metadata) const
{
    assert(output != nullptr);

    if (std::to_underlying(condition & (Glib::IOCondition::IO_IN | Glib::IOCondition::IO_HUP)) != 0) {
        stream_metadata->append_line_to_buffer(output);
        return true;
    }

    /*
     * If we've triggered the callback with something other than a IOCondition::IO_IN or IOCondition::IO_HUP,
     * something unexpected has happened and Glib is indicating an error state.
     */
    logger->warn("Abnormal IO condition reported by GLib for sub-process stream: code " +
            std::to_string(std::to_underlying(condition)) + '.');
    return false;
}

void StreamingProcessExecutor::reap_child(const Glib::Pid ended_pid, const int exit_code) noexcept
{
    if (ended_pid != pid)
        return;

    /*
     * We would like to keep to RAII as much as practicable, but the streams must be irrevocably reset upon
     * sub-process termination otherwise the entire thread will hang.
```

```
     */
    stdout_stream.reset();
    stderr_stream.reset();

    // Call this last, as it invokes a synchronous user-provided callback that may require disconnected
    // streams.
    ProcessExecutor::reap_child(ended_pid, exit_code);
}

Glib::RefPtr<Gtk::TextTag> StreamingProcessExecutor::get_colour_tag(
        const Glib::RefPtr<Gtk::TextBuffer> &output, const Glib::ustring &name,
        const Glib::ustring &colour_name)
{
    auto tag(output->get_tag_table()->lookup(name));

    if (tag == nullptr) {
        tag = output->create_tag(name);
        tag->property_foreground().set_value(colour_name);
    }

    return tag;
}

StreamingProcessExecutor::Stream::Stream(const Glib::RefPtr<Gtk::TextTag> &formatting_tag, int source_fd,
        sigc::bound_mem_functor<decltype(&StreamingProcessExecutor::stream_callback), Glib::IOCondition,
                const Stream *> &&write_line_callback) :
    formatting_tag(formatting_tag),
    channel(Glib::IOChannel::create_from_fd(source_fd)),
    bound_write_line_callback(sigc::bind(write_line_callback, this)),
    watch(Glib::signal_io().connect(bound_write_line_callback, channel,
            Glib::IOCondition::IO_IN | Glib::IOCondition::IO_HUP | Glib::IOCondition::IO_ERR))
{
}

StreamingProcessExecutor::Stream::~Stream()
{
    try {
        /*
         * Even with the buffers flushed, we manually execute the callback just to be safe. In the worst case,
         * there's nothing to read. We explicitly ignore the return code of the callback, as during
         * destruction we don't care if the channel is reporting a HUP.
         */
        if (channel->flush() == Glib::IOStatus::NORMAL)
            std::ignore = bound_write_line_callback(Glib::IOCondition::IO_IN);

        watch.disconnect();
        channel->close();
    } catch (const Glib::IOChannelError &channel_error) {
        logger->error("Could not graciously destruct stream for sub-process.");
        logger->error(channel_error.what());
    }
}

void StreamingProcessExecutor::Stream::append_line_to_buffer(
        const Glib::RefPtr<Gtk::TextBuffer> &target_buffer) const
{
    Glib::ustring content;

    // Read to the end to receive all content since the last read. There is no need to delimit on line
    // boundaries.
    if (channel->read_to_end(content) == Glib::IOStatus::NORMAL)
        target_buffer->insert_with_tag(target_buffer->end(), content, formatting_tag);
}

} // namespace optifol
```

1.8.13 StreamingProcessExecutor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the external streaming Process Executor
 * @author Oliver Dixon
 * @date 2025-07-27
 * @version Development
```

```
 */

#ifndef STREAMINGPROCESSEXECUTOR_HPP
#define STREAMINGPROCESSEXECUTOR_HPP

#include <glibmm/iochannel.h>
#include <gtkmm/textbuffer.h>

#include "ProcessExecutor.hpp"

namespace optifol
{

/**
 * @class StreamingProcessExecutor
 * @brief Provides a ProcessExecutor with the capability to capture <code>stdout</code> and
 *  <code>stderr</code> streams from the sub-process and write the colour-formatted lines to a
 *  Gtk::TextBuffer.
 * @see ProcessExecutor for semantics of external sub-process management.
 */
class StreamingProcessExecutor : public ProcessExecutor
{
public:
    /**
     * @brief Asynchronously execute an external process at the given working directory, with the given
     * argument vector, optionally including a key-value environment specification and exit callback.
     * @param working_directory The working directory in which the sub-process should be started.
     * @param argv The argument vector, where the first entry is the command to execute. The system PATH will
     * be searched to find the executable, but it is not run under a shell.
     * @param envp The optional <code>KEY=VALUE</code> set of environment variables to export into the
     * sub-process environment.
     * @param output The destination output buffer to which the sub-process <code>stdout</code> and
     * <code>stderr</code> lines should be redirected.
     * @param finished_callback The noexcept callback to execute on completion of the sub-process, taking the
     * numerical exit code.
     */
    StreamingProcessExecutor(const std::string &working_directory, const std::vector<std::string> &argv,
            const std::vector<std::string> &envp, const Glib::RefPtr<Gtk::TextBuffer> &output,
            sigc::slot<void(int)> &&finished_callback);

    /**
     * @brief Statically write a Glib::SpawnError exception message, formatted as an error, to the given
     * output.
     * @param exception The exception to describe on the output buffer.
     * @param output The Gtk::TextBuffer to receive the human-readable exception message.
     */
    static void write_exception_error(
            const Glib::SpawnError &exception, const Glib::RefPtr<Gtk::TextBuffer> &output);

private:
    static const log4cxx::LoggerPtr logger;

    class Stream;

    /**
     * @brief Callback to indicate that new data has arrived at the given stream.
     * @param condition The condition under which the data has arrived.
     * @param stream_metadata The stream object encapsulating the incoming data.
     * @pre The @ref output buffer is not null.
     * @return Was the data received under a non-erroneous condition?
     */
    bool stream_callback(Glib::IOCondition condition, const Stream *stream_metadata) const;

    /**
     * @copybrief ProcessExecutor::reap_child
     * @details In addition to reaping the child process and calling the user-supplied callback, the streams
     * are first disconnected to prevent thread hangs.
     * @param ended_pid The PID of the ended process.
     * @param exit_code The numerical exit code of the process; anything except zero is deemed indicative of
     * an error.
     */
    void reap_child(Glib::Pid ended_pid, int exit_code) noexcept override;

    /**
     * @brief Get or create a formatted colour tag on the given output buffer.
     * @param output The output to which format-tagged lines should be written.
     * @param name The internal name of the GTK tag.
     * @param colour_name The name of the colour to use, e.g. "red".
     * @return The formatting tag.
     */
    static Glib::RefPtr<Gtk::TextTag> get_colour_tag(const Glib::RefPtr<Gtk::TextBuffer> &output,
```

```
            const Glib::ustring &name, const Glib::ustring &colour_name);

    /**
     * @class Stream
     * @brief Helper for watching a single stream and despatching a callback on new lines of data.
     */
    class Stream
    {
    public:
        /**
         * @brief Construct a new Stream to watch the data on the given file descriptor
         * @param formatting_tag The @ref Gtk::TextTag containing formatting information for data produced by
         * the Stream
         * @param source_fd The numerical file descriptor of the Stream to watch
         * @param write_line_callback The callback to execute when new data becomes available on the Stream
         */
        explicit Stream(const Glib::RefPtr<Gtk::TextTag> &formatting_tag, int source_fd,
                sigc::bound_mem_functor<decltype(&StreamingProcessExecutor::stream_callback),
                        Glib::IOCondition, const Stream *> &&write_line_callback);

        /**
         * @brief Destruct the Stream by flushing the buffers, invoking any callbacks, and disconnecting
         * watches and signals.
         */
        ~Stream();

        /**
         * @brief Append the newest line of data on the Stream to the given buffer using the fixed formatting
         * tag
         * @param target_buffer The target buffer to receive the latest Stream line of data
         */
        void append_line_to_buffer(const Glib::RefPtr<Gtk::TextBuffer> &target_buffer) const;

    private:
        Glib::RefPtr<Gtk::TextTag> formatting_tag;
        const Glib::RefPtr<Glib::IOChannel> channel;
        sigc::slot<bool(Glib::IOCondition)> bound_write_line_callback;
        sigc::connection watch;
    };

    const Glib::RefPtr<Gtk::TextBuffer> output;
    std::optional<Stream> stdout_stream;
    std::optional<Stream> stderr_stream;
};

} // namespace optifol

#endif // STREAMINGPROCESSEXECUTOR_HPP
```

1.8.14 AnalysisArea

1.8.14.1 AnalysisArea.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Analysis and Optimisation view in the Optifol Main Window
 * @author Oliver Dixon
 * @date 2025-04-27
 * @version Development
 */

#include "AnalysisArea.hpp"

#include "../../Storage/Subsystem.hpp"
#include "../GTKHelpers.hpp"
#include "../RequirementsIndexArea/RequirementsIndexArea.hpp"

namespace optifol
{

void AnalysisArea::add_query_page()
{
    auto &new_page =
            query_pages.emplace_back("new query", get_selected_analysis_group()->observe_prover_instance());
```

```
    new_page.add_to_notebook(*queries_notebook);
}

const char *const AnalysisArea::area_name = "Analysis and Optimisation Area";

AnalysisArea::AnalysisArea(Gtk::Builder &builder) :
    on_off_widgets(GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "analysis_advice_unselected"),
            GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "analysis_index_content")),
    groups_view(GTKHelpers::get_widget<Gtk::ColumnView>(area_name, builder, "analysis_groups_view")),
    context_menu(groups_view,
            GTKHelpers::get_object<Gio::Menu>(area_name, builder, "analysis_groups_context_menu"),
            {{"new_analysis_group",
                     GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "new_analysis_group"),
                     GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "new_analysis_group_popover"),
                     true},
                    {"edit_analysis_group",
                            GTKHelpers::get_widget<Gtk::MenuButton>(
                                    area_name, builder, "edit_analysis_group"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "edit_analysis_group_popover"),
                            false},
                    {"delete_analysis_group",
                            GTKHelpers::get_widget<Gtk::MenuButton>(
                                    area_name, builder, "delete_analysis_group"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "delete_analysis_group_popover"),
                            false}}),
    new_analysis_group_popover(builder, *this),
    queries_notebook(GTKHelpers::get_widget<Gtk::Notebook>(area_name, builder, "analysis_queries"))
{
    selection_model->set_autoselect(false);
    selection_model->set_can_unselect(true);

    selection_model->signal_selection_changed().connect(
            [this](guint, const guint n_items)
            {
                if (n_items == 0 || selection_model->get_selected() == 0) {
                    context_menu.disable_action("edit_analysis_group");
                    context_menu.disable_action("delete_analysis_group");
                } else {
                    context_menu.enable_action("edit_analysis_group");
                    context_menu.enable_action("delete_analysis_group");
                }
            });

    selection_model->signal_items_changed().connect(
            [this](const guint added, const guint removed, guint)
            {
                if (added > 0)
                    add_query_page();

                if (removed > 0) {
                    // If anything was removed from the model, just deselect everything out of an abundance of
                    // caution.
                    context_menu.disable_action("edit_analysis_group");
                    context_menu.disable_action("delete_analysis_group");
                }
            });

    groups_view->set_model(selection_model);

    const auto columns = groups_view->get_columns();
    const auto column_count = columns->get_n_items();

    for (guint position = 0; position < column_count; ++position) {
        Glib::RefPtr<Gtk::ColumnViewColumn> column = nullptr;

        if ((column = columns->get_typed_object<Gtk::ColumnViewColumn>(position)) != nullptr) {
            const auto &gtk_id = column->get_id();
            const auto factory = Gtk::SignalListItemFactory::create();

            if (gtk_id == "analysis_requirement_name") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_expandable_label, false));
                factory->signal_bind().connect(
                        [this](const Glib::RefPtr<Gtk::ListItem> &list_item)
                        {
                            /*
                             * Cannot use sigc::bind on non-constant fixed arguments, as any fixed argument
                             * values are evaluated on execution of the bind call. Hence, calling
                             * bind(tree_model) on the below function would fix all subsequent invocations to
```

```
                             * the initial value of the tree_model.
                             */
                            StorageObjectBase::bind_name_property_expandable(list_item, tree_model);
                        });

            } else if (gtk_id == "analysis_requirement_statement") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, true));
                factory->signal_bind().connect(
                        sigc::ptr_fun(&RequirementsIndexArea::on_bind_property_statement));

            } else if (gtk_id == "analysis_requirement_cnf") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, true));
                factory->signal_bind().connect(
                        sigc::ptr_fun(&RequirementsIndexArea::on_bind_property_normalised));

            } else

                // Jump out here if unrecognised, so all further code can assume a factory was configured.
                continue;

            column->set_factory(factory);
        }
    }

    queries_notebook->signal_page_added().connect(
            [this](Gtk::Widget *const, const guint) noexcept { queries_notebook->set_visible(); });

    queries_notebook->signal_page_removed().connect(
            [this](Gtk::Widget *const, const guint) noexcept
            {
                if (queries_notebook->get_n_pages() == 0)
                    queries_notebook->set_visible(false);
            });
}

AnalysisArea::~AnalysisArea()
{
    assert(queries_notebook->get_n_pages() == query_pages.size());

    if (queries_notebook->get_n_pages() > 1) {
        const auto page_count = static_cast<guint>(query_pages.size());
        for (guint page_index = page_count - 1; page_index > 1; --page_index) {
            queries_notebook->remove_page(static_cast<gint>(page_index));
            query_pages.pop_back();
        }
    }

    if (queries_notebook->get_n_pages() == 1) {
        queries_notebook->remove_page(0);
        query_pages.pop_back();
    }

    assert(queries_notebook->get_n_pages() == 0 && query_pages.empty());
}

void AnalysisArea::select_model(const Glib::RefPtr<Subsystem> &subsystem_model)
{
    on_off_widgets.first->set_visible(false);
    on_off_widgets.second->set_visible(true);

    active_subsystem = subsystem_model;
    tree_model = Gtk::TreeListModel::create(
            active_subsystem->get_analysis_groups(), &ObjectGroup<Requirement>::get_model_callback, true);
    selection_model->set_model(tree_model);
}

void AnalysisArea::deselect_model()
{
    on_off_widgets.second->set_visible(false);
    on_off_widgets.first->set_visible(true);

    active_subsystem = nullptr;
    tree_model = nullptr;
    selection_model->set_model(nullptr);
}

Subsystem *AnalysisArea::get_active_subsystem() noexcept
{
    return active_subsystem.get();
```

```
}

const Subsystem *AnalysisArea::observe_active_subsystem() const noexcept
{
    return active_subsystem.get();
}

Glib::RefPtr<AnalysisGroup> AnalysisArea::get_selected_analysis_group() const
{
    auto selected_row = tree_model->get_row(selection_model->get_selected());
    if (selected_row == nullptr)
        throw std::runtime_error("Popover was made available despite no suitable Analysis Group selection.");

    while (selected_row->get_depth() > 0)
        selected_row = selected_row->get_parent();

    const auto analysis_group = std::dynamic_pointer_cast<AnalysisGroup>(selected_row->get_item());
    if (analysis_group == nullptr)
        throw std::runtime_error("Popover could not find a suitable Analysis Group.");

    return analysis_group;
}

} // namespace optifol
```

1.8.14.2 AnalysisArea.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Analysis and Optimisation view in the Optifol Main Window
 * @author Oliver Dixon
 * @date 2025-04-27
 * @version Development
 */

#ifndef ANALYSISAREA_HPP
#define ANALYSISAREA_HPP

#include <gtkmm/builder.h>
#include <gtkmm/columnview.h>
#include <gtkmm/notebook.h>
#include <gtkmm/singleselection.h>
#include <gtkmm/treelistmodel.h>

#include "../../Storage/AnalysisGroup.hpp"
#include "../ContextButtonCorrespondence.hpp"
#include "../IWindowArea.hpp"
#include "AnalysisAreaNewAnalysisGroupPopover.hpp"
#include "AnalysisQuery.hpp"

namespace optifol
{

/**
 * @class AnalysisArea
 * @brief Manage the <i>Analysis and Optimisation</i> area
 *
 * @details
 *  <p>
 *      The <i>Analysis and Optimisation</i> area provides controls for categorising Requirement objects from
 * the RequirementsIndexArea into AnalysisGroup groups. Groups may then be individually subject to
 * mathematical analysis under the FOL engine, with the results of analysis displayed in a tabbed paned view.
 * The following GTK elements are expected from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th>
 *              <th>Unique Identifier</th>
 *              <th>Purpose</th>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Widget (abstract)</td>
 *              <td><code>analysis_advice_unselected</code></td>
 *              <td>Advice to display when the area is unavailable</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Widget (abstract)</td>
 *              <td><code>analysis_index_content</code></td>
```

```
*             <td>Replacement to <code>analysis_advice_unselected</code>, containing all active content</td>
*         </tr>
*         <tr>
*             <td>Gtk::ColumnView</td>
*             <td><code>analysis_groups_view</code></td>
*             <td>Table containing AnalysisGroup and child Requirement entries</td>
*         </tr>
*         <tr>
*             <td>Gio::Menu</td>
*             <td><code>analysis_groups_context_menu</code></td>
*             <td>Area-wide context menu</td>
*         </tr>
*         <tr>
*             <td>Gtk::MenuButton</td>
*             <td><code>new_analysis_group</code></td>
*             <td>Button for creating a New AnalysisGroup</td>
*         </tr>
*         <tr>
*             <td>Gtk::Popover</td>
*             <td><code>new_analysis_group_popover</code></td>
*             <td>Popover UI for creating a New AnalysisGroup</td>
*         </tr>
*         <tr>
*             <td>Gtk::MenuButton</td>
*             <td><code>edit_analysis_group</code></td>
*             <td>Button for creating a editing an existing AnalysisGroup</td>
*         </tr>
*         <tr>
*             <td>Gtk::Popover</td>
*             <td><code>edit_analysis_group_popover</code></td>
*             <td>Popover UI for editing an existing AnalysisGroup</td>
*         </tr>
*         <tr>
*             <td>Gtk::MenuButton</td>
*             <td><code>delete_analysis_group</code></td>
*             <td>Button for creating a deleting an existing AnalysisGroup</td>
*         </tr>
*         <tr>
*             <td>Gtk::Popover</td>
*             <td><code>delete_analysis_group_popover</code></td>
*             <td>Popover UI for deleting an existing AnalysisGroup</td>
*         </tr>
*         <tr>
*             <td>Gtk::MenuButton</td>
*             <td><code>create_new_query</code></td>
*             <td>Button for creating a new query</td>
*         </tr>
*         <tr>
*             <td>Gtk::MenuButton</td>
*             <td><code>delete_selected_query</code></td>
*             <td>Button for deleting the selected query</td>
*         </tr>
*         <tr>
*             <td>Gtk::Notebook</td>
*             <td><code>analysis_queries</code></td>
*             <td>Notebook containing the executed queries for the selected AnalysisGroup</td>
*         </tr>
*         <tr>
*             <td>Gtk::ColumnViewColumn</td>
*             <td><code>analysis_requirement_name</code></td>
*             <td>Table column for the Requirement / AnalysisGroup name</td>
*         </tr>
*         <tr>
*             <td>Gtk::ColumnViewColumn</td>
*             <td><code>analysis_requirement_statement</code></td>
*             <td>Table column for the Requirement statement</td>
*         </tr>
*         <tr>
*             <td>Gtk::ColumnViewColumn</td>
*             <td><code>analysis_requirement_cnf</code></td>
*             <td>Table column for the Requirement statement in normalised CNF</td>
*         </tr>
*     </table>
*     A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in
* the expected type instantiations.
*  </p>
*  <p>
*     In addition to the stated required GTK elements, constituent popovers of this view will require their
* own, possibly distinct, set of elements: <ul> <li>@ref AnalysisAreaNewAnalysisGroupPopover</li>
*     </ul>
*  </p>
```

```
 */
class AnalysisArea : public IWindowArea
{
public:
    /**
     * @brief Construct a new compartmentalised area for displaying and managing sets of subsystem
     * requirements
     * @param builder The GTK builder attached to the main window
     */
    explicit AnalysisArea(Gtk::Builder &builder);

    ~AnalysisArea() override;

    void select_model(const Glib::RefPtr<Subsystem> &subsystem_model) override;

    void deselect_model() override;

    Subsystem *get_active_subsystem() noexcept override;

    const Subsystem *observe_active_subsystem() const noexcept override;

private:
    Glib::RefPtr<AnalysisGroup> get_selected_analysis_group() const;

    void add_query_page();

    static const char *const area_name;

    Glib::RefPtr<Subsystem> active_subsystem;
    Glib::RefPtr<Gtk::SingleSelection> selection_model = Gtk::SingleSelection::create();
    Glib::RefPtr<Gtk::TreeListModel> tree_model;

    std::pair<Gtk::Widget *, Gtk::Widget *> on_off_widgets;
    std::vector<AnalysisQuery> query_pages;

    Gtk::ColumnView *groups_view;
    Gtk::Notebook *queries_notebook;

    ContextButtonCorrespondence context_menu;

    AnalysisAreaNewAnalysisGroupPopover new_analysis_group_popover;
};

} // namespace optifol

#endif
```

1.8.14.3 AnalysisAreaNewAnalysisGroupPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Analysis Area "New Analysis Group" popover manager
 * @author Oliver Dixon
 * @date 2025-07-09
 * @version Development
 */

#include "AnalysisAreaNewAnalysisGroupPopover.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "../Storage/AnalysisGroup.hpp"
#include "../Storage/Subsystem.hpp"
#include "AnalysisArea.hpp"

namespace optifol
{

const char *const AnalysisAreaNewAnalysisGroupPopover::popover_name = "New Analysis Group Popover";
const log4cxx::LoggerPtr AnalysisAreaNewAnalysisGroupPopover::popover_logger =
        Logging::get_logger({"GUI", "AnalysisOptimisation", "NewAnalysisGroup"});

AnalysisAreaNewAnalysisGroupPopover::AnalysisAreaNewAnalysisGroupPopover(
        Gtk::Builder &builder, AnalysisArea &analysis_area) :
    analysis_area(analysis_area),
```

```
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "new_analysis_group_popover")),
    confirm_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "new_analysis_group_confirm")),
    cancel_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "new_analysis_group_cancel")),
    name_entry(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "new_analysis_group_property_name"))
{
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &AnalysisAreaNewAnalysisGroupPopover::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &AnalysisAreaNewAnalysisGroupPopover::cancel_button_clicked));
}

void AnalysisAreaNewAnalysisGroupPopover::confirm_button_clicked() const
{
    my_popover->popdown();
    analysis_area.observe_active_subsystem()->get_analysis_groups()->append(
            Glib::make_refptr_for_instance(new AnalysisGroup(name_entry->get_text(),
                    analysis_area.get_active_subsystem()->share_symbol_repository())));

    popover_logger->debug("Created new analysis group with name \"" + name_entry->get_text() + "\".");
    clear_inputs();
}

void AnalysisAreaNewAnalysisGroupPopover::cancel_button_clicked() const
{
    my_popover->popdown();
    clear_inputs();
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from button-click callback.
void AnalysisAreaNewAnalysisGroupPopover::clear_inputs() const
{
    name_entry->set_text("");
}

} // namespace optifol
```

1.8.14.4 AnalysisAreaNewAnalysisGroupPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Analysis Area "New Analysis Group" popover manager
 * @author Oliver Dixon
 * @date 2025-07-09
 * @version Development
 */

#ifndef ANALYSISAREANEWANALYSISGROUPPOPOVER_HPP
#define ANALYSISAREANEWANALYSISGROUPPOPOVER_HPP

#include <gtkmm/builder.h>
#include <gtkmm/button.h>
#include <gtkmm/entry.h>
#include <gtkmm/popover.h>
#include <log4cxx/logger.h>

namespace optifol
{

class AnalysisArea;

/**
 * @class AnalysisAreaNewAnalysisGroupPopover
 * @brief Manage the <i>New Analysis Group</i> popover for the Analysis area
 * @see AnalysisArea for the parent area
 *
 * @details
 *  The <i>New Analysis Group</i> popover provides controls for creating a new AnalysisGroup, consisting of
 * one or more Requirement objects from the active Subsystem, to be subject to mathematical analysis. The
 * following GTK elements are expected from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th>
 *          <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
```

```
 *          <td><code>new_analysis_group_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>new_analysis_group_confirm</code></td>
 *          <td>Confirm creation of a new AnalysisGroup</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>new_analysis_group_cancel</code></td>
 *          <td>Cancels creation of a new AnalysisGroup</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>new_analysis_group_property_name</code></td>
 *          <td>Text entry area for the name of the new AnalysisGroup</td>
 *      </tr>
 *  </table>
 *  A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in the
 * expected type instantiations.
 */
class AnalysisAreaNewAnalysisGroupPopover : public sigc::trackable
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param analysis_area A mutating reference to the view of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    AnalysisAreaNewAnalysisGroupPopover(Gtk::Builder &builder, AnalysisArea &analysis_area);

private:
    /**
     * @brief Handle a click of the <i>Confirm</i> by attempting to create the named AnalysisGroup.
     */
    void confirm_button_clicked() const;

    /**
     * @brief Handle a click of the <i>Cancel</i> button by discarding all input and closing the popover.
     */
    void cancel_button_clicked() const;

    /**
     * @brief Clear all user fields in the popover
     */
    void clear_inputs() const;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    AnalysisArea &analysis_area;
    Gtk::Popover *const my_popover;
    Gtk::Button *const confirm_button;
    Gtk::Button *const cancel_button;
    Gtk::Entry *const name_entry;
};

} // namespace optifol

#endif // ANALYSISAREANEWANALYSISGROUPPOPOVER_HPP
```

1.8.14.5 AnalysisQueryCanvas.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the proof trace canvas
 * @author Oliver Dixon
 * @date 2025-02-03
 * @version Development
 */

#include "AnalysisQueryCanvas.hpp"
```

```cpp
#include <cassert>
#include <ranges>

namespace optifol
{

using namespace CanvasSupport;

AnalysisQueryCanvas::AnalysisQueryCanvas()
{
    set_draw_func(sigc::mem_fun(*this, &AnalysisQueryCanvas::on_draw));
}

void AnalysisQueryCanvas::replace_proof(const ProofTreeNode *terminating_node)
{
    assert(terminating_node->observe_node()->get_triviality_state() == Clause::State::TriviallyFalse);
    add_node(terminating_node);
}

const NodeDrawingAdapter *AnalysisQueryCanvas::add_node(const ProofTreeNode *node)
{
    const NodeDrawingAdapter *lhs_adapter = nullptr;
    const NodeDrawingAdapter *rhs_adapter = nullptr;

    if (node->get_depth() > 0) {
        if (node->observe_lhs_parent() != nullptr)
            lhs_adapter = add_node(node->observe_lhs_parent());

        if (node->observe_rhs_parent() != nullptr)
            rhs_adapter = add_node(node->observe_rhs_parent());
    }

    return &nodes[node->get_depth()].emplace_back(node, lhs_adapter, rhs_adapter);
}

void AnalysisQueryCanvas::on_draw(const Cairo::RefPtr<Cairo::Context> &ctx, const int width, const int height)
{
    std::ignore = width;
    std::ignore = height;

    if (nodes.empty())
        return;

    Cairo::FontExtents font_extents;

    ctx->select_font_face("Monospace", Cairo::ToyFontFace::Slant::NORMAL, Cairo::ToyFontFace::Weight::NORMAL);
    ctx->set_font_size(14.0);
    ctx->get_font_extents(font_extents);

    float last_x_pos = 0;

    // Draw edges and prepare the node coordinates and Cairo text extents.
    for (auto &[depth, layer_contents]: nodes)
        for (auto &&node: layer_contents)
            last_x_pos = draw_edge(*ctx, node, last_x_pos, font_extents);

    int maximised_width = 0;
    int maximised_height = 0;

    // Draw the first layer of nodes (from which the north-eastern-most point can be deduced).
    for (const auto &node: nodes.begin()->second) {
        const auto end_point = draw_proof_node(*ctx, node);
        maximised_width = std::max(maximised_width, static_cast<int>(end_point.x));
    }

    // Draw the middle section of nodes.
    for (const auto &layer: nodes | std::views::drop(1) | std::views::take(nodes.size() - 2))
        for (const auto &node: layer.second)
            draw_proof_node(*ctx, node);

    // Draw the final layer of nodes (from which the south-eastern-most point can be deduced).
    const auto &last_layer_it = --nodes.end();
    if (last_layer_it != nodes.begin())
        for (const auto &node: last_layer_it->second) {
            const auto end_point = draw_proof_node(*ctx, node);
            maximised_height = std::max(maximised_height, static_cast<int>(end_point.y));
        }

    content_width = maximised_width + static_cast<int>(h_spacing);
    content_height = maximised_height + static_cast<int>(v_spacing);
```

```
    set_size_request(content_width, content_height);
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive.
Point AnalysisQueryCanvas::draw_proof_node(Cairo::Context &ctx, const NodeDrawingAdapter &node)
{
    assert(node.start.x < node.centre.x);
    assert(node.start.y < node.centre.y);
    assert(node.label_extents.width > 0);
    assert(node.label_extents.height > 0);

    // Draw the rectangle body, coloured according to its position in the proof trace.
    if (node.node->get_depth() == 0)
        axiom_node_colour.apply(ctx);
    else if (node.node->observe_node()->get_triviality_state() == Clause::State::TriviallyFalse)
        terminating_node_colour.apply(ctx);
    else
        deduction_node_colour.apply(ctx);

    // The rectangle requires space to house the text, plus padding along all sides.
    const float rect_w = static_cast<float>(node.label_extents.width) + 2 * rectangle_padding;
    const float rect_h = static_cast<float>(node.label_extents.height) + 2 * rectangle_padding;

    ctx.rectangle(node.start.x, node.start.y, rect_w, rect_h);
    ctx.fill();

    // Draw the rectangle border.
    border_colour.apply(ctx);
    ctx.rectangle(node.start.x, node.start.y, rect_w, rect_h);
    ctx.stroke();

    // Draw the text inside the rectangle.
    text_colour.apply(ctx);
    ctx.move_to(node.start.x - node.label_extents.x_bearing + rectangle_padding,
            node.start.y - node.label_extents.y_bearing + rectangle_padding);

    ctx.show_text(node.node_label);

    // Report the south-eastern-most point drawn.
    return {node.centre.x + rect_w / 2.0f, node.centre.y + rect_h / 2.0f};
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive.
void AnalysisQueryCanvas::draw_unifier(Cairo::Context &ctx, const NodeDrawingAdapter &node,
        const LineSegment &lhs_edge, const LineSegment &rhs_edge, const Cairo::FontExtents &font_extents)
{
    assert(node.start.x < node.centre.x);
    assert(node.start.y < node.centre.y);

    /*
     * Draw the unifier text entries (each representing a substitution made in the resolution clause) centred
     * above the resolvent, aligned horizontally with the average of the distances to either parent.
     */
    const auto lhs_avg = (node.centre.y + node.lhs->centre.y) / 2;
    const auto rhs_avg = (node.centre.y + node.rhs->centre.y) / 2;

    const auto [lhs_midpoint, rhs_midpoint] = draw_unifier_entries(
            ctx, node.edge_label_lines, Point(node.centre.x, (lhs_avg + rhs_avg) / 2), font_extents);

    /*
     * Compute the endpoints of the visible edges, chopping off endpoints that are hidden behind nodes. This
     * is required for computing the midpoint of the hypotenuse that connects the resolvent with each parent.
     */
    const LineSegment lhs_intercepting_segment{
            lhs_edge.trace(node.lhs->centre.y + (node.lhs->centre.y - node.lhs->start.y)),
            lhs_edge.trace(node.start.y)};

    const LineSegment rhs_intercepting_segment{
            rhs_edge.trace(node.rhs->centre.y + (node.rhs->centre.y - node.rhs->start.y)),
            rhs_edge.trace(node.start.y)};

    // Draw lines between the midpoints of the visible parent-child edges and the centre-aligned unifier line
    // block.

    unifier_edge_colour.apply(ctx);
    LineSegment(lhs_intercepting_segment.get_midpoint(), lhs_midpoint).draw(ctx);
    LineSegment(rhs_intercepting_segment.get_midpoint(), rhs_midpoint).draw(ctx);
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive.
float AnalysisQueryCanvas::draw_edge(Cairo::Context &ctx, CanvasSupport::NodeDrawingAdapter &node,
```

```
        float last_x_pos, const Cairo::FontExtents &font_extents)
{
    // Compute the coordinates and the text extents of the node.
    ctx.get_text_extents(node.node_label, node.label_extents);

    const auto depth = node.node->get_depth();
    node.start.y = v_spacing * (static_cast<float>(depth) + 1);
    node.centre.y = node.start.y + static_cast<float>(node.label_extents.height) / 2.0f + rectangle_padding;

    if (depth > 0) {
        assert(node.lhs != nullptr);
        assert(node.rhs != nullptr);
        node.centre.x =
                node.lhs->centre.x + (node.rhs->centre.x - node.lhs->centre.x) / 2 + rectangle_padding;
    } else {
        node.centre.x =
                last_x_pos + h_spacing + static_cast<float>(node.label_extents.width) / 2 + rectangle_padding;
        last_x_pos = node.centre.x + static_cast<float>(node.label_extents.width) / 2;
    }

    node.start.x = node.centre.x - static_cast<float>(node.label_extents.width) / 2;

    if (depth > 0) {
        /*
         * If we're not drawing an axiom (i.e. a resolvent) there will be two edges to each parent (LHS and
         * RHS). There may also be an applicable multi-line unifier text block to render at the midpoint,
         * which will require the resolvent to be moved down proportional to the number of unification
         * entries.
         */
        const float advance_y =
                static_cast<float>(node.edge_label_lines.size()) * static_cast<float>(font_extents.height) +
                2.0f * rectangle_padding;
        node.start.y += advance_y;
        node.centre.y += advance_y;

        // Draw the edges from each parent to the resolvent.
        const LineSegment lhs_edge(node.centre, node.lhs->centre);
        const LineSegment rhs_edge(node.centre, node.rhs->centre);

        edge_colour.apply(ctx);
        lhs_edge.draw(ctx);
        rhs_edge.draw(ctx);

        if (!node.edge_label_lines.empty())
            // If applicable, draw the unifier and connecting lines.
            draw_unifier(ctx, node, lhs_edge, rhs_edge, font_extents);
    }

    return last_x_pos;
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive.
std::pair<Point, Point> AnalysisQueryCanvas::draw_unifier_entries(Cairo::Context &ctx,
        const std::vector<std::string> &unifier_lines, const Point &centre,
        const Cairo::FontExtents &font_extents)
{
    // Determine the box width according to the maximum line length of the rendered unifier entries.
    float max_width = 0;
    for (const auto &line: unifier_lines) {
        Cairo::TextExtents line_extents;
        ctx.get_text_extents(line, line_extents);
        max_width = std::max(max_width, static_cast<float>(line_extents.width));
    }

    const float box_width = max_width + 2 * rectangle_padding;
    const float box_height =
            static_cast<float>(unifier_lines.size()) * static_cast<float>(font_extents.height) +
            2 * rectangle_padding;

    const Point start{
            static_cast<float>(centre.x - box_width / 2.0), static_cast<float>(centre.y - box_height / 2.0)};

    // Write lines inside the box, incrementing the cursor by the baseline skip extent.
    text_colour.apply(ctx);
    float cursor_y = start.y + rectangle_padding + static_cast<float>(font_extents.ascent);
    for (const auto &line: unifier_lines) {
        ctx.move_to(start.x + rectangle_padding, cursor_y);
        ctx.show_text(line);
        cursor_y += static_cast<float>(font_extents.height);
    }
```

```
    // Report the midpoints of the vertical bounding lines.
    return {Point(start.x, centre.y), Point(start.x + box_width, centre.y)};
}

Gtk::SizeRequestMode AnalysisQueryCanvas::get_request_mode_vfunc() const
{
    return Gtk::SizeRequestMode::CONSTANT_SIZE;
}

void AnalysisQueryCanvas::measure_vfunc(const Gtk::Orientation orientation, const int for_size, int &minimum,
        int &natural, int &minimum_baseline, int &natural_baseline) const
{
    if (orientation == Gtk::Orientation::HORIZONTAL) {
        minimum = content_width;
        natural = content_width;
    } else {
        minimum = content_height;
        natural = content_height;
    }
}

} // namespace optifol
```

1.8.14.6 AnalysisQueryCanvas.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the proof trace canvas
 * @author Oliver Dixon
 * @date 2025-02-03
 * @version Development
 */

#ifndef OPTIFOL_ANALYSISQUERYCANVAS_HPP
#define OPTIFOL_ANALYSISQUERYCANVAS_HPP

#include <gtkmm/drawingarea.h>

#include "../../Inference/Resolvent.hpp"
#include "CanvasSupport.hpp"

namespace optifol
{

/**
 * @class AnalysisQueryCanvas
 * @brief The canvas for rendering resolution proof trees with Cairo.
 */
class AnalysisQueryCanvas : public Gtk::DrawingArea
{
public:
    /**
     * @brief Construct an empty canvas.
     */
    AnalysisQueryCanvas();

    /**
     * @brief Replace any existing proofs with a new proof, terminating by the given node in the tree.
     * @param terminating_node The derived contradiction, representing the deepest element in the execution
     * trace.
     */
    void replace_proof(const ProofTreeNode *terminating_node);

    Gtk::SizeRequestMode get_request_mode_vfunc() const override;

    void measure_vfunc(Gtk::Orientation orientation, int for_size, int &minimum, int &natural,
            int &minimum_baseline, int &natural_baseline) const override;

private:
    /**
     * @brief Recursively wrap the given ProofTreeNode (and its parents) in drawing adapters and add to the
     * store.
     * @param node The root node to add.
     * @return An observing pointer to the registered root node.
```

```
     */
    const CanvasSupport::NodeDrawingAdapter *add_node(const ProofTreeNode *node);

    /**
     * @brief Redraws a visualisation of the current proof trace to the given Cairo::Context.
     * @param ctx The Cairo::Context to which the proof trace render should be drawn.
     * @param width Not used, required by the GTKmm interface.
     * @param height Not used, required by the GTKmm interface.
     */
    void on_draw(const Cairo::RefPtr<Cairo::Context> &ctx, int width, int height);

    /**
     * @brief Draw a single ProofTreeNode encapsulated by the given NodeDrawingAdapter.
     * @details For any type of ProofTreeNode, this renderer draws a bordered box containing a single
     *  monospace-formatted text representing the serialisation of the node. This represents a Clause in the
     * proof.
     * @param ctx The Cairo::Context to which the node should be drawn.
     * @param node The NodeDrawingAdapter containing the node to be drawn.
     * @return The Point at the south-eastern-most (i.e. most extreme) point drawn by the function.
     * @pre The X coordinate of the NodeDrawingAdapter has been properly initialised.
     * @pre The Y coordinate of the NodeDrawingAdapter has been properly initialised.
     * @pre The text extents of the label has been properly initialised.
     */
    static CanvasSupport::Point draw_proof_node(
            Cairo::Context &ctx, const CanvasSupport::NodeDrawingAdapter &node);

    /**
     * @brief Render a unifier on an edge, with adjoining lines indicating the source Clauses and Resolvent.
     * @param ctx The Cairo::Context to which the unifier and lines should be drawn.
     * @param node The NodeDrawingAdapter encapsulating the Resolvent with the Unifier to be drawn.
     * @param lhs_edge The LineSegment joining the centre of the Resolvent node with the centre of its LHS
     * parent.
     * @param rhs_edge The LineSegment joining the centre of the Resolvent node with the centre of its RHS
     * parent.
     * @param font_extents The precomputed extents of the font, required for baseline skip calculations.
     * @pre The X coordinate of the NodeDrawingAdapter has been properly initialised.
     * @pre The Y coordinate of the NodeDrawingAdapter has been properly initialised.
     */
    static void draw_unifier(Cairo::Context &ctx, const CanvasSupport::NodeDrawingAdapter &node,
            const CanvasSupport::LineSegment &lhs_edge, const CanvasSupport::LineSegment &rhs_edge,
            const Cairo::FontExtents &font_extents);

    /**
     * @brief Prepares a node, and, if applicable, draws its parent-child edges and unifier.
     * @param ctx The Cairo::Context to which the edge and unifier should be drawn.
     * @param node The node to be prepared.
     * @param last_x_pos The trailing X position of the last node at the same depth.
     * @param font_extents The precomputed extents of the font, required for baseline skip calculations.
     * @return The trailing X position of the rendered node.
     */
    static float draw_edge(Cairo::Context &ctx, CanvasSupport::NodeDrawingAdapter &node, float last_x_pos,
            const Cairo::FontExtents &font_extents);

    /**
     * @brief Render the unifier entries, with one substitution per line, at the given centre point.
     * @param ctx The Cairo::Context to which the text should be rendered.
     * @param unifier_lines The serialised unifier lines.
     * @param centre The centre point of the bounding box in which the text is rendered.
     * @param font_extents The precomputed extents of the font, required for baseline skip calculations.
     * @return The midpoints of the LHS and RHS vertical boundaries of the bounding box.
     */
    static std::pair<CanvasSupport::Point, CanvasSupport::Point> draw_unifier_entries(Cairo::Context &ctx,
            const std::vector<std::string> &unifier_lines, const CanvasSupport::Point &centre,
            const Cairo::FontExtents &font_extents);

    static constexpr float rectangle_padding = 6;
    static constexpr float h_spacing = 100;
    static constexpr float v_spacing = 80;

    std::map<unsigned int, std::deque<CanvasSupport::NodeDrawingAdapter>> nodes;

    int content_width = 0;
    int content_height = 0;
};

} // namespace optifol

#endif // OPTIFOL_ANALYSISQUERYCANVAS_HPP
```

1.8.14.7 AnalysisQuery.cpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Analysis Query GTK page
 * @author Oliver Dixon
 * @date 2026-01-25
 * @version Development
 */

#include "AnalysisQuery.hpp"

#include <cassert>

#include "../../Exceptions/SemanticException.hpp"

namespace optifol
{

std::istringstream AnalysisQuery::lexer_input_stream;

FOLLexer AnalysisQuery::lexer{AnalysisQuery::lexer_input_stream, std::cerr};
FOLParser AnalysisQuery::parser{&AnalysisQuery::lexer};

AnalysisQuery::AnalysisQuery(const Glib::ustring &query_name, Prover &kb_weak) :
    kb_weak(kb_weak)
{
    execute_query_button.signal_clicked().connect(sigc::mem_fun(*this, &AnalysisQuery::execute_query));
    tab_label.set_text(query_name);

    // The inner box contains the query entry area
    query_entry.set_placeholder_text("Query");
    query_entry.set_hexpand();

    execute_query_button.set_icon_name("edit-find-symbolic");
    execute_query_button.set_tooltip_text("Execute Query");

    inner_box.set_hexpand();
    inner_box.add_css_class("optifol_boxed");
    inner_box.append(query_entry);
    inner_box.append(execute_query_button);

    // The scrolled window encapsulates the Cairo drawing surface
    drawing_area.set_hexpand(false);
    drawing_area.set_vexpand(false);
    scrolled_window.set_vexpand();
    scrolled_window.set_child(drawing_area);

    // The outer box is vertical-oriented and stacks the query entry on top of the scrollable drawing area.
    outer_box.set_orientation(Gtk::Orientation::VERTICAL);
    outer_box.append(inner_box);
    outer_box.append(scrolled_window);
}

void AnalysisQuery::add_to_notebook(Gtk::Notebook &notebook)
{
    notebook.append_page(outer_box, tab_label);
}

void AnalysisQuery::execute_query()
{
    if (query_entry.get_text().empty())
        return;

    lexer_input_stream.str(query_entry.get_text());
    parser.parse();

    latest_result = std::make_unique<QueryResult>(kb_weak.ask(parser.retrieve_sentence()));
    if (latest_result->terminating_resolvent != nullptr)
        drawing_area.replace_proof(latest_result->terminating_resolvent);

    drawing_area.queue_draw();
}

} // namespace optifol
```

1.8.14.8 AnalysisQuery.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Analysis Query GTK page
 * @author Oliver Dixon
 * @date 2026-01-25
 * @version Development
 */

#ifndef OPTIFOL_ANALYSISQUERY_HPP
#define OPTIFOL_ANALYSISQUERY_HPP

#include <gtkmm/box.h>
#include <gtkmm/button.h>
#include <gtkmm/entry.h>
#include <gtkmm/label.h>
#include <gtkmm/notebook.h>
#include <gtkmm/scrolledwindow.h>

#include "../../Inference/Prover.hpp"
#include "../../Inference/QueryResult.hpp"
#include "AnalysisQueryCanvas.hpp"
#include "FOLLexer.hpp"

namespace optifol
{

/**
 * @class AnalysisQuery
 * @brief A single page in a Gtk::Notebook holding the tab widget Gtk::Label and the scrollable Gtk::TextView.
 * @warning The Gtk::Notebook widget does not own its child pages; to avoid memory access problems, always
 * remove pages from the Gtk::Notebook before destructing the corresponding AnalysisQuery.
 */
class AnalysisQuery
{
public:
    /**
     * @brief Create a new page for another query on the KB.
     * @param query_name The human-readable string of the query.
     * @param kb_weak A weak reference to the prover instance encapsulating the KB to be queried.
     */
    explicit AnalysisQuery(const Glib::ustring &query_name, Prover &kb_weak);

    /**
     * @brief Append the built page to an arbitrary notebook.
     * @param notebook The notebook to which the page should be appended.
     */
    void add_to_notebook(Gtk::Notebook &notebook);

private:
    /**
     * @brief Check consistency of the current query (in the text box) with the KB. If a contradiction is
     * derived, a proof trace is generated in the drawing area.
     */
    void execute_query();

    std::vector<std::unique_ptr<Literal>> literal_store;

    Gtk::Label tab_label;
    Gtk::ScrolledWindow scrolled_window;
    AnalysisQueryCanvas drawing_area;
    Gtk::Box outer_box;
    Gtk::Box inner_box;
    Gtk::Entry query_entry;
    Gtk::Button execute_query_button;

    Prover &kb_weak;
    std::unique_ptr<QueryResult> latest_result;

    static std::istringstream lexer_input_stream;
    static FOLLexer lexer;
    static FOLParser parser;
};
```

```
} // namespace optifol

#endif // OPTIFOL_ANALYSISQUERY_HPP
```

1.8.14.9 CanvasSupport.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Collection of graphics-related helper structs for the AnalysisQueryCanvas
 * @author Oliver Dixon
 * @date 2025-02-03
 * @version Development
 */

#ifndef OPTIFOL_CANVASSUPPORT_HPP
#define OPTIFOL_CANVASSUPPORT_HPP

#include <cairomm/context.h>
#include <cassert>

#include "../../Inference/ProofTreeNode.hpp"

namespace optifol::CanvasSupport
{

/**
 * @struct NodeColour
 * @brief A POD RGB colour representation that can be applied to a Cairo::Context.
 */
struct NodeColour
{

    /**
     * @brief The red component, normalised in the range [0, 1].
     */
    float red;

    /**
     * @brief The green component, normalised in the range [0, 1].
     */
    float green;

    /**
     * @brief The green component, normalised in the range [0, 1].
     */
    float blue;

    /**
     * @brief Set the given Cairo::Context to use the colour.
     * @param ctx The target Cairo::Context.
     */
    void apply(Cairo::Context &ctx) const
    {
        ctx.set_source_rgb(red, green, blue);
    }
};

constexpr NodeColour axiom_node_colour{234.0f / 255, 207.0f / 255, 193.0f / 255};
constexpr NodeColour terminating_node_colour{178.0f / 255, 210.0f / 255, 226.0f / 255};
constexpr NodeColour deduction_node_colour{0.95f, 0.95f, 0.95f};
constexpr NodeColour text_colour{0.0f, 0.0f, 0.0f};
constexpr NodeColour border_colour{0.2f, 0.2f, 0.2f};
constexpr NodeColour edge_colour{0.0f, 0.0f, 0.0f};
constexpr NodeColour unifier_edge_colour{0.7f, 0.7f, 0.7f};

/**
 * @struct CanvasDrawable
 * @brief A type that is drawable to a Cairo::Context canvas.
 */
struct CanvasDrawable
{
    /**
     * @brief Destruct the CanvasDrawable.
     */
```

```
    virtual ~CanvasDrawable() = default;

    /**
     * @brief Draw the CanvasDrawable item in the colour currently selected by the Cairo::Context.
     * @param ctx The target canvas.
     */
    virtual void draw(Cairo::Context &ctx) const = 0;

    /**
     * @brief Draw the CanvasDrawable item in the given colour.
     * @param ctx The target canvas.
     * @param node_colour The normalised RGB colour.
     */
    void draw(Cairo::Context &ctx, const NodeColour &node_colour) const
    {
        node_colour.apply(ctx);
        draw(ctx);
    }
};

/**
 * @struct Point
 * @brief A CanvasDrawable two-dimensional point on a Cairo::Context canvas.
 */
struct Point : CanvasDrawable
{
    /**
     * @brief Construct the origin point.
     */
    Point() = default;

    /**
     * @brief Construct the point at the given coordinates.
     * @param x The X coordinate.
     * @param y The Y coordinate.
     */
    Point(const float x, const float y) :
        x(x),
        y(y)
    {
    }

    void draw(Cairo::Context &ctx) const override
    {
        ctx.arc(x, y, 5, 0, 2 * M_PI);
        ctx.stroke();
    }

    float x = 0;
    float y = 0;
};

/**
 * @struct LineSegment
 * @brief A line segment (a line with two fixed endpoints) on a Cairo::Context canvas.
 */
struct LineSegment : CanvasDrawable
{
    /**
     * @brief Construct the line segment with the given endpoints.
     * @param p1 The first endpoint.
     * @param p2 The second endpoint.
     */
    LineSegment(const Point &p1, const Point &p2) :
        endpoint_1(p1),
        endpoint_2(p2),
        gradient(p1.x == p2.x ? INFINITY : (p1.y - p2.y) / (p1.x - p2.x))
    {
    }

    /**
     * @brief Determine the X coordinate on the non-horizontal line for the given Y coordinate.
     * @param y The Y position on the line for which the corresponding X is sought.
     * @throws std::runtime_error The line was horizontal, hence there is no unique X for a given Y.
     * @return The Point on the LineSegment at the given Y position.
     * @note This function does not check that the returned pair is actually on the LineSegment. The API user
     * should verify that the given Y is within the bounds of the LineSegment endpoints.
     */
    [[nodiscard]] Point trace(const float y) const
    {
        switch (std::fpclassify(gradient)) {
```

```
        case FP_INFINITE:
            // The line is vertical.
            return {endpoint_1.x, y};
        case FP_ZERO:
            // The line is horizontal.
            throw std::runtime_error(
                    "There is no unique X coordinate for a given Y on a horizontal line segment.");
        default:
            return {endpoint_1.x + (y - endpoint_1.y) / gradient, y};
        }
    }

    /**
     * @brief Gets the midpoint of the line.
     * @return The midpoint of the LineSegment.
     */
    [[nodiscard]] Point get_midpoint() const noexcept
    {
        return {(endpoint_1.x + endpoint_2.x) / 2, (endpoint_1.y + endpoint_2.y) / 2};
    }

    void draw(Cairo::Context &ctx) const override
    {
        ctx.move_to(endpoint_1.x, endpoint_1.y);
        ctx.line_to(endpoint_2.x, endpoint_2.y);
        ctx.stroke();
    }

    const Point endpoint_1;
    const Point endpoint_2;
    const float gradient;
};

/**
 * @struct NodeDrawingAdapter
 * @brief A transparent cache of drawing information related to a ProofTreeNode, for direct use by
 * AnalysisQueryCanvas.
 */
struct NodeDrawingAdapter
{
    /**
     * @brief Construct a new adapter for the given node, with optional LHS and RHS parents.
     * @param node The self ProofTreeNode, typically a Resolvent, but sometimes a Clause (for axioms in the
     * proof).
     * @param lhs An optional parent on the LHS.
     * @param rhs An optional parent on the RHS.
     * @note The given node must have exactly zero or exactly two parents; this is enforced at runtime.
     */
    NodeDrawingAdapter(const ProofTreeNode *const node, const NodeDrawingAdapter *const lhs,
            const NodeDrawingAdapter *const rhs) :
        node_label(generate_label(*node)),
        node(node),
        lhs(lhs),
        rhs(rhs)
    {
        assert((!lhs && !rhs) || (lhs && rhs));

        if (node->observe_edge().has_value())
            // If applicable, cache the delimeted lines in the unifier text block entry.
            edge_label_lines = std::move(node->observe_edge().value()->split_serialise());
    }

    Point start;
    Point centre;
    Cairo::TextExtents label_extents{};

    const std::string node_label;
    std::vector<std::string> edge_label_lines;
    const ProofTreeNode *node;
    const NodeDrawingAdapter *lhs = nullptr;
    const NodeDrawingAdapter *rhs = nullptr;

private:
    /**
     * @brief Stringify the given ISerialisable.
     * @param node The node to serialise.
     * @return The serialised string produced from the node.
     */
    static std::string generate_label(const ISerialisable &node)
    {
        std::ostringstream oss;
```

```
        oss << node;
        return oss.str();
    }
};

} // namespace optifol::CanvasSupport

#endif // OPTIFOL_CANVASSUPPORT_HPP
```

1.8.15 ProjectHierarchyPane

1.8.15.1 ProjectHierarchyPane.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Project Hierarchical view in the Optifol Main Window
 * @author Oliver Dixon
 * @date 2025-03-27
 * @version Development
 */

#include "ProjectHierarchyPane.hpp"

#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "../Storage/Subsystem.hpp"

namespace optifol
{

std::shared_ptr<log4cxx::Logger> ProjectHierarchyPane::logger(log4cxx::Logger::getLogger("Optifol"));

const char *const ProjectHierarchyPane::area_name = "Project Pane Area";

ProjectHierarchyPane::ProjectHierarchyPane(
        Gtk::Builder &builder, const Glib::RefPtr<Gio::ListStore<Project>> &initial_model) :
    stack_switcher(GTKHelpers::get_widget<Gtk::DropDown>(area_name, builder, "project_pane_switcher")),
    stack(GTKHelpers::get_widget<Gtk::Stack>(area_name, builder, "project_pane_stack")),
    view(GTKHelpers::get_widget<Gtk::ListView>(area_name, builder, "project_view")),
    data_model(initial_model),
    tree_model(Gtk::TreeListModel::create(
            data_model, sigc::ptr_fun(&ProjectHierarchyPane::tree_node_expand), true, true)),
    context_menu(view, GTKHelpers::get_object<Gio::Menu>(area_name, builder, "structure_context_menu"),
            {{"new_project", GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "new_project"),
                     GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "new_project_popover"), true},
                    {"new_subsystem",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "new_subsystem"),
                            GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "new_subsystem_popover"),
                            false},
                    {"edit_structure",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "edit_structure"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "edit_structure_popover"),
                            false},
                    {"delete_structure",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "delete_structure"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "delete_structure_popover"),
                            false}})
{
    view->signal_activate().connect(sigc::mem_fun(*this, &ProjectHierarchyPane::switch_selection));

    stack_switcher->property_selected().signal_changed().connect(
            sigc::mem_fun(*this, &ProjectHierarchyPane::switch_visible_stack));

    selection_model = Gtk::SingleSelection::create(tree_model);
    selection_model->set_autoselect(false);
    selection_model->set_can_unselect(true);
    view->set_model(selection_model);

    selection_model->signal_selection_changed().connect(
            [this](guint, const guint n_items)
            {
                if (n_items == 0) {
```

```
                emit_deselected();
                context_menu.disable_action("new_subsystem");
                context_menu.disable_action("edit_structure");
                context_menu.disable_action("delete_structure");
            } else {
                context_menu.enable_action("new_subsystem");
                context_menu.enable_action("edit_structure");
                context_menu.enable_action("delete_structure");
            }
        });

    selection_model->signal_items_changed().connect(
        [this](guint, const guint removed, guint)
        {
            if (removed > 0) {
                // If anything was removed from the model, just deselect everything out of an abundance of
                // caution.
                emit_deselected();
                context_menu.disable_action("new_subsystem");
                context_menu.disable_action("edit_structure");
                context_menu.disable_action("delete_structure");
            }
        });

    const auto factory = Gtk::SignalListItemFactory::create();
    factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_expandable_label, false));
    factory->signal_bind().connect(sigc::bind(&StorageObjectBase::bind_name_property_expandable, tree_model));
    view->set_factory(factory);

    configure_new_project_popover(builder);
    configure_new_subsystem_popover(builder);
    configure_edit_structure_popover(builder);
    configure_delete_structure_popover(builder);
}

void ProjectHierarchyPane::add_subsystem_change_callback(
        sigc::slot<void(const Glib::RefPtr<Subsystem> &)> &&selected, sigc::slot<void()> &&deselected,
        const bool onboard)
{
    auto &callback = subsystem_change_callbacks.emplace_back();
    callback.first.connect(std::move(selected));
    callback.second.connect(std::move(deselected));

    if (onboard) {
        const auto subsystem = std::dynamic_pointer_cast<Subsystem>(selection_model->get_selected_item());
        if (subsystem != nullptr)
            callback.first(subsystem);
        else
            callback.second();
    }
}

void ProjectHierarchyPane::configure_new_project_popover(Gtk::Builder &builder) const
{
    const auto popover = GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "new_project_popover");
    const auto confirm_button =
            GTKHelpers::get_widget<Gtk::Button>(area_name, builder, "new_project_confirm");
    const auto cancel_button = GTKHelpers::get_widget<Gtk::Button>(area_name, builder, "new_project_cancel");
    const auto property_name =
            GTKHelpers::get_widget<Gtk::Entry>(area_name, builder, "new_project_property_name");

    cancel_button->signal_clicked().connect(
            [popover, property_name]
            {
                popover->popdown();
                property_name->set_text("");
            });

    confirm_button->signal_clicked().connect(
            [this, popover, property_name]
            {
                popover->popdown();
                data_model->append(Glib::make_refptr_for_instance(new Project(property_name->get_text())));
                property_name->set_text("");
            });
}

void ProjectHierarchyPane::configure_new_subsystem_popover(Gtk::Builder &builder) const
{
    const auto popover = GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "new_subsystem_popover");
    const auto confirm_button =
```

```
        GTKHelpers::get_widget<Gtk::Button>(area_name, builder, "new_subsystem_confirm");
    const auto cancel_button =
        GTKHelpers::get_widget<Gtk::Button>(area_name, builder, "new_subsystem_cancel");
    const auto property_path =
        GTKHelpers::get_widget<Gtk::Entry>(area_name, builder, "new_subsystem_property_path");
    const auto property_name =
        GTKHelpers::get_widget<Gtk::Entry>(area_name, builder, "new_subsystem_property_name");

    popover->signal_show().connect(
        [this, property_path]
        {
            const auto candidate =
                std::dynamic_pointer_cast<const TreeNode>(selection_model->get_selected_item());

            if (candidate != nullptr)
                property_path->set_text(candidate->get_path());
        });

    cancel_button->signal_clicked().connect(
        [popover, property_name]
        {
            popover->popdown();
            property_name->set_text("");
        });

    confirm_button->signal_clicked().connect(
        [this, popover, property_name]
        {
            popover->popdown();

            const auto candidate =
                std::dynamic_pointer_cast<TreeNode>(selection_model->get_selected_item());
            if (candidate != nullptr)
                candidate->add(property_name->get_text());

            property_name->set_text("");
        });
}

void ProjectHierarchyPane::configure_edit_structure_popover(Gtk::Builder &builder) const
{
    const auto popover = GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "edit_structure_popover");
    const auto confirm_button =
        GTKHelpers::get_widget<Gtk::Button>(area_name, builder, "edit_structure_confirm");
    const auto cancel_button =
        GTKHelpers::get_widget<Gtk::Button>(area_name, builder, "edit_structure_cancel");
    const auto property_old_path =
        GTKHelpers::get_widget<Gtk::Entry>(area_name, builder, "edit_structure_property_old_path");
    const auto property_new_path =
        GTKHelpers::get_widget<Gtk::Entry>(area_name, builder, "edit_structure_property_new_path");

    popover->signal_show().connect(
        [this, property_old_path, property_new_path]
        {
            const auto candidate =
                std::dynamic_pointer_cast<const TreeNode>(selection_model->get_selected_item());

            if (candidate != nullptr) {
                const auto current_path = candidate->get_path();
                property_old_path->set_text(current_path);

                const auto name_delim_position = current_path.rfind('/');
                property_new_path->set_text(name_delim_position == std::string::npos
                                ? current_path
                                : current_path.substr(0, name_delim_position + 1));
            }
        });

    cancel_button->signal_clicked().connect([popover] { popover->popdown(); });

    confirm_button->signal_clicked().connect(
        [this, popover]
        {
            popover->popdown();
        });
}

void ProjectHierarchyPane::configure_delete_structure_popover(Gtk::Builder &builder) const
{
    const auto popover = GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "delete_structure_popover");
    const auto confirm_button =
```

```
            GTKHelpers::get_widget<Gtk::Button>(area_name, builder, "delete_structure_confirm");
    const auto cancel_button =
            GTKHelpers::get_widget<Gtk::Button>(area_name, builder, "delete_structure_cancel");
    const auto property_path =
            GTKHelpers::get_widget<Gtk::Entry>(area_name, builder, "delete_structure_property_path");

    popover->signal_show().connect(
            [this, property_path]
            {
                const auto candidate =
                        std::dynamic_pointer_cast<const TreeNode>(selection_model->get_selected_item());
                if (candidate != nullptr)
                    property_path->set_text(candidate->get_path() + "/*");
            });

    cancel_button->signal_clicked().connect([popover] { popover->popdown(); });

    confirm_button->signal_clicked().connect(
            [this, popover]
            {
                popover->popdown();

                const auto candidate =
                        std::dynamic_pointer_cast<TreeNode>(selection_model->get_selected_item());

                if (candidate != nullptr) {
                    const auto parent = candidate->get_parent();
                    if (parent != nullptr) {
                        // If an owning model is explicitly declared, use that.
                        const auto owning_model = parent->get_children();
                        const auto owning_model_count = owning_model->get_n_items();
                        for (guint idx = 0; idx < owning_model_count; ++idx)
                            if (owning_model->get_item(idx) == candidate)
                                owning_model->remove(idx);
                    } else {
                        // Otherwise, use the root model.
                        const auto root_model_count = data_model->get_n_items();
                        for (guint idx = 0; idx < root_model_count; ++idx)
                            if (data_model->get_item(idx) == candidate)
                                data_model->remove(idx);
                    }
                }

                emit_deselected();
            });
}

void ProjectHierarchyPane::tree_node_setup(const Glib::RefPtr<Gtk::ListItem> &item)
{
    const auto expander = Gtk::make_managed<Gtk::TreeExpander>();
    const auto label = Gtk::make_managed<Gtk::Label>();

    label->set_halign(Gtk::Align::START);
    expander->set_child(*label);
    item->set_child(*expander);
}

void ProjectHierarchyPane::switch_visible_stack() const
{
    switch (stack_switcher->get_selected()) {
    case static_cast<guint>(ProjectStackSwitcherIdx::Explorer):
        stack->set_visible_child("project_explorer");
        break;
    case static_cast<guint>(ProjectStackSwitcherIdx::Metadata):
        stack->set_visible_child("project_metadata");
        break;
    default:
        break;
    }
}

Glib::RefPtr<Gio::ListModel> ProjectHierarchyPane::tree_node_expand(
        const Glib::RefPtr<Glib::ObjectBase> &item)
{
    const auto candidate = std::dynamic_pointer_cast<TreeNode>(item);

    if (candidate != nullptr)
        return candidate->get_children();

    return nullptr;
}
```

```
// ReSharper disable once CppDFAUnreachableFunctionCall - false positive
void ProjectHierarchyPane::emit_selected(const Glib::RefPtr<Subsystem> &new_subsystem) const
{
    for (const auto &[select_callback, _]: subsystem_change_callbacks)
        select_callback.emit(new_subsystem);
}

void ProjectHierarchyPane::emit_deselected() const
{
    for (const auto &[_, deselect_callback]: subsystem_change_callbacks)
        deselect_callback.emit();
}

void ProjectHierarchyPane::switch_selection(guint) const
{
    // Rely on the selection model to inform on the selected item. The view can be unreliable.
    const auto subsystem = std::dynamic_pointer_cast<Subsystem>(selection_model->get_selected_item());
    if (subsystem != nullptr)
        emit_selected(subsystem);
    else
        emit_deselected();
}

} // namespace optifol
```

1.8.15.2 ProjectHierarchyPane.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Project Hierarchical view in the Optifol Main Window
 * @author Oliver Dixon
 * @date 2025-03-27
 * @version Development
 */

#ifndef PROJECTHIERARCHYPANE_HPP
#define PROJECTHIERARCHYPANE_HPP

#include <gtkmm.h>
#include <log4cxx/logger.h>

#include "../../Storage/Project.hpp"
#include "../AnalysisArea/AnalysisArea.hpp"
#include "../ContextButtonCorrespondence.hpp"

namespace optifol
{

/**
 * @class ProjectHierarchyPane
 * @brief The manager of the UI view element representing an interactable tree of all projects and subsystems,
 * typically derived from a backend storage controller.
 */
class ProjectHierarchyPane
{
public:
    /**
     * @brief Construct a project pane manager given a managed Gtk::ListView resource and backend model
     * @param builder The GTK builder attached to the main window
     * @param initial_model The backend storage model used to populate the model and stream data updates
     */
    ProjectHierarchyPane(Gtk::Builder &builder, const Glib::RefPtr<Gio::ListStore<Project>> &initial_model);

    /**
     * @brief Add a new listener whose callback routine should be called when a new Subsystem is selected, or
     * the previously selected one is deselected.
     * @param selected The callback to handle the changing of the active selection to a new Subsystem
     * @param deselected The callback to handle the deselection of the currently selected Subsystem
     * @param onboard Should the newly added callback be invoked immediately to "onboard" the listener as to
     * the current state of the selection? The deselection callback is called during onboarding if and only if
     * the present selection is invalid or not a Subsystem.
     */
    void add_subsystem_change_callback(sigc::slot<void(const Glib::RefPtr<Subsystem> &)> &&selected,
```

```
            sigc::slot<void()> &&deselected, bool onboard = true);

private:
    enum class ProjectStackSwitcherIdx
    {
        Explorer = 0,
        Metadata = 1
    };

    /**
     * @brief Configure the sub-widgets of the 'New Project' popover
     * @param builder The builder associated with the popover
     */
    void configure_new_project_popover(Gtk::Builder &builder) const;

    /**
     * @brief Configure the sub-widgets of the 'New Subsystem' popover
     * @param builder The builder associated with the popover
     */
    void configure_new_subsystem_popover(Gtk::Builder &builder) const;

    /**
     * @brief Configure the sub-widgets of the 'Edit Structure' popover
     * @param builder The builder associated with the popover
     */
    void configure_edit_structure_popover(Gtk::Builder &builder) const;

    /**
     * @brief Configure the sub-widgets of the 'Delete Structure' popover
     * @param builder The builder associated with the popover
     */
    void configure_delete_structure_popover(Gtk::Builder &builder) const;

    /**
     * @brief GTK callback for a new Gtk::ListItem. This member function handles the configuration of a new
     * entry in the Project Hierarchy view as an editable label.
     * @param item The new list item to configure for placement within the tree view
     */
    static void tree_node_setup(const Glib::RefPtr<Gtk::ListItem> &item);

    /**
     * @brief GTK callback for a change of selection on the stack-switcher dropdown. The current state is
     * checked, and the corresponding stack page is made visible.
     */
    void switch_visible_stack() const;

    /**
     * @brief Retrieves sub-models for tree items being expanded
     * @param item The item being expanded
     * @return The model representing the children of the expanded object
     */
    static Glib::RefPtr<Gio::ListModel> tree_node_expand(const Glib::RefPtr<Glib::ObjectBase> &item);

    /**
     * @brief Emits a notification to all listeners that a new subsystem has been selected
     */
    void emit_selected(const Glib::RefPtr<Subsystem> &new_subsystem) const;

    /**
     * @brief Emits a notification to all listeners that the previously selected subsystem has been
     * deselected.
     */
    void emit_deselected() const;

    /**
     * @brief Handle a selection change in the Project Hierarchy Pane by updating any internal state and
     * informing listeners
     */
    void switch_selection(guint) const;

    static const char *const area_name;
    static std::shared_ptr<log4cxx::Logger> logger;

    std::vector<std::pair<sigc::signal<void(const Glib::RefPtr<Subsystem> &)>, sigc::signal<void()>>>
            subsystem_change_callbacks;

    Gtk::DropDown *const stack_switcher;
    Gtk::Stack *const stack;
    Gtk::ListView *const view;

    Glib::RefPtr<Gio::ListStore<Project>> data_model;
```

```
    Glib::RefPtr<Gtk::SingleSelection> selection_model;
    const Glib::RefPtr<Gtk::TreeListModel> tree_model;

    ContextButtonCorrespondence context_menu;
};

} // namespace optifol

#endif
```

1.8.16 ReportsArea

1.8.16.1 ReportsArea.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Main Window's Releases and Reports area
 * @author Oliver Dixon
 * @date 2025-07-08
 * @version Development
 */

#include "ReportsArea.hpp"
#include "../GTKHelpers.hpp"

namespace optifol
{

const char *const ReportsArea::area_name = "Releases and Reports Area";

ReportsArea::ReportsArea(Gtk::Builder &builder) :
    on_off_widgets(GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "reports_advice_unselected"),
            GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "reports_content")),
    view(GTKHelpers::get_widget<Gtk::ColumnView>(area_name, builder, "reports_elements_view")),
    context_menu(view, GTKHelpers::get_object<Gio::Menu>(area_name, builder, "reports_context_menu"),
            {{"reports_generate_latex",
                    GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "reports_generate_latex"),
                    GTKHelpers::get_widget<Gtk::Popover>(
                            area_name, builder, "reports_generate_latex_popover"),
                    true}}),
    generate_latex_popover(builder, *this)
{
}

void ReportsArea::select_model(const Glib::RefPtr<Subsystem> &subsystem)
{
    on_off_widgets.first->set_visible(false);
    on_off_widgets.second->set_visible(true);
    active_subsystem = subsystem;
}

void ReportsArea::deselect_model()
{
    on_off_widgets.second->set_visible(false);
    on_off_widgets.first->set_visible(true);
    active_subsystem = nullptr;
}

Subsystem *ReportsArea::get_active_subsystem() noexcept
{
    return active_subsystem.get();
}

const Subsystem *ReportsArea::observe_active_subsystem() const noexcept
{
    return active_subsystem.get();
}

} // namespace optifol
```

1.8.16.2 ReportsAreaGenerateLaTeXPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Reports Area "Generate LaTeX" popover manager
 * @author Oliver Dixon
 * @date 2025-07-08
 * @version Development
 */

#include <fstream>

#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "ReportsArea.hpp"
#include "ReportsAreaGenerateLaTeXPopover.hpp"

namespace optifol
{

const char *const ReportsAreaGenerateLaTeXPopover::popover_name = "Generate LaTeX Report Popover";
const log4cxx::LoggerPtr ReportsAreaGenerateLaTeXPopover::popover_logger =
        Logging::get_logger({"GUI", "ReleasesReports", "GenerateLaTeX"});

ReportsAreaGenerateLaTeXPopover::ReportsAreaGenerateLaTeXPopover(
        Gtk::Builder &builder, const ReportsArea &reports_area) :
    reports_area(reports_area),
    buffer(GTKHelpers::get_widget<Gtk::TextView>(popover_name, builder, "generate_latex_output")
                   ->get_buffer()),
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "reports_generate_latex_popover")),
    details_container(
            GTKHelpers::get_widget<Gtk::Box>(popover_name, builder, "generate_latex_details_container")),
    output_directory_entry(
            GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "generate_latex_output_path")),
    confirm_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "generate_latex_confirm")),
    cancel_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "generate_latex_cancel")),
    open_dialog(GTKHelpers::get_object<Gtk::FileDialog>(
            popover_name, builder, "generate_latex_output_path_chooser_dialog")),
    show_details_check(
            GTKHelpers::get_widget<Gtk::CheckButton>(popover_name, builder, "generate_latex_show_details"))
{

    show_details_check->signal_toggled().connect(
            sigc::mem_fun(*this, &ReportsAreaGenerateLaTeXPopover::show_details_toggled));
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &ReportsAreaGenerateLaTeXPopover::confirm_button_clicked));

    const auto open_directory_button = GTKHelpers::get_widget<Gtk::Button>(
            popover_name, builder, "generate_latex_output_path_chooser_button");
    open_directory_button->signal_clicked().connect(
            sigc::mem_fun(*this, &ReportsAreaGenerateLaTeXPopover::open_directory_button_clicked));
}

void ReportsAreaGenerateLaTeXPopover::confirm_button_clicked()
{
    confirm_button->set_sensitive(false);
    buffer->set_text("");

    /*
     * Create a new generator instance (possibly replacing the previous one) to produce a file in the given
     * output directory indicated by the user. Populate it with the Requirements, the Test Groups, and the
     * Analysis Groups, and despatch the asynchronous generation of the report with a callback to run on
     * completion.
     */
    generator.emplace(output_directory);

    // Populate the Requirements Index.
    reports_area.observe_active_subsystem()->for_each(
            [this](const Requirement &requirement) { generator->add_requirement(requirement); });

    // Populate the Test Groups.
    const auto test_groups = reports_area.observe_active_subsystem()->get_test_groups();
    const auto test_group_count = test_groups->get_n_items();
    for (guint test_group_index = 0; test_group_index < test_group_count; ++test_group_index)
        generator->add_test_group(*test_groups->get_item(test_group_index));

    // Run the generator.
    generator->generate(
```

```
            buffer, sigc::mem_fun(*this, &ReportsAreaGenerateLaTeXPopover::post_generation_callback));
}

void ReportsAreaGenerateLaTeXPopover::cancel_button_clicked()
{
    my_popover->popdown();
    clear_inputs();
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from button-click callback.
void ReportsAreaGenerateLaTeXPopover::clear_inputs()
{
    output_directory_entry->set_text("");
    buffer->set_text("");
    output_directory = nullptr;
}

void ReportsAreaGenerateLaTeXPopover::open_directory_button_clicked()
{
    // Hide the popover whilst the dialog is active, otherwise it may have z-index priority over the dialog.
    my_popover->popdown();
    open_dialog->select_folder(
            sigc::mem_fun(*this, &ReportsAreaGenerateLaTeXPopover::open_directory_finished));
}

void ReportsAreaGenerateLaTeXPopover::open_directory_finished(const Glib::RefPtr<Gio::AsyncResult> &result)
{
    try {
        output_directory = open_dialog->select_folder_finish(result);
    } catch (const Gtk::DialogError &dialog_error) {
        popover_logger->info("Directory selection dialog closed without feedback; state unchanged: " +
                std::string(dialog_error.what()));
    } catch (const Glib::Error &library_error) {
        popover_logger->error("Unexpected error from directory selection dialog; state unchanged: " +
                std::string(library_error.what()));
    }

    // Restore the temporarily hidden popover
    my_popover->popup();

    // Always update the read-only text entry with the up-to-date output directory path
    output_directory_entry->set_text(output_directory == nullptr ? "" : output_directory->get_path());
}

void ReportsAreaGenerateLaTeXPopover::post_generation_callback() const
{
    confirm_button->set_sensitive();
}

void ReportsAreaGenerateLaTeXPopover::show_details_toggled() const
{
    details_container->set_visible(show_details_check->get_active());
}

} // namespace optifol
```

1.8.16.3 ReportsAreaGenerateLaTeXPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Reports Area "Generate LaTeX" popover manager
 * @author Oliver Dixon
 * @date 2025-07-08
 * @version Development
 */

#ifndef REPORTSAREAGENERATELATEXPOPOVER_HPP
#define REPORTSAREAGENERATELATEXPOPOVER_HPP

#include <glibmm/refptr.h>
#include <gtkmm/box.h>
#include <gtkmm/builder.h>
#include <gtkmm/button.h>
#include <gtkmm/checkbutton.h>
```

```
#include <gtkmm/entry.h>
#include <gtkmm/filedialog.h>
#include <gtkmm/popover.h>
#include <gtkmm/textbuffer.h>
#include <log4cxx/logger.h>

#include "../../Reporting/LaTeXReportGenerator.hpp"

namespace optifol
{

class ReportsArea;

/**
 * @class ReportsAreaGenerateLaTeXPopover
 * @brief Manage the "Generate LaTeX" popover for the Releases and Reports Area
 * @note Instantiations of this class mutate the graphical environment via the given Gtk::Builder. As such, it
 * should be regarded an effective singleton as multiple instantiations will cause conflicts with the GTK+
 * runtime due to multiply registered callbacks.
 * @see ReportsArea for parent area
 *
 * @details
 *  <p>
 *      The "Generate LaTeX" popover provides controls for setting LaTeX-specific options in the automated
 * report- generation process and invoking the external commands to construct a LaTeX and subsequent PDF
 * document. The following GTK elements are expected from the given Gtk::Builder: <table> <tr> <th>GTK C++
 * Class</th> <th>Unique Identifier</th> <th>Purpose</th>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Popover</td>
 *              <td><code>reports_generate_latex_popover</code></td>
 *              <td>Managed popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::TextView</td>
 *              <td><code>generate_latex_output</code></td>
 *              <td>Output view of latexmk command</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Button</td>
 *              <td><code>generate_latex_confirm</code></td>
 *              <td><i>Confirm</i> button to generate LaTeX and PDF</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Button</td>
 *              <td><code>generate_latex_cancel</code></td>
 *              <td><i>Cancel</i> button to discard process</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Entry</td>
 *              <td><code>generate_latex_output_path</code></td>
 *              <td>Entry field to display destination directory for LaTeX artefacts and produced PDF</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Button</td>
 *              <td><code>generate_latex_output_path_chooser_button</code></td>
 *              <td>Button to launch directory/file chooser for output destination</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::CheckButton</td>
 *              <td><code>generate_latex_show_details</code></td>
 *              <td>Checkbox to enable or disable display of latexmk compiler output</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::FileDialog</td>
 *              <td><code>generate_latex_output_path_chooser_dialog</code></td>
 *              <td>File/directory chooser dialog for selecting output destination</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Box</td>
 *              <td><code>generate_latex_details_container</code></td>
 *              <td>Container including compiler output and some visual separators</td>
 *          </tr>
 *      </table>
 *      A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in
 * the expected type instantiations.
 *  </p>
 *  <p>
 *      The implementation and state machine of this class is complicated by the process of despatching and
 * watching the output buffers (via numerical file descriptors for <code>stdout</code> and
 * <code>stderr</code>) of external processes; in particular, the <code>latexmk</code> utility used to build
```

```
 * PDFs from LaTeX (<code>.tex</code>) source files. The implementation here is believed to be reasonably
 * fool-proof, albeit specialised to the single use-case, and elegantly handles data from both standard output
 * streams, errors, and HUPs indicating process completion.
 *  </p>
 */
class ReportsAreaGenerateLaTeXPopover : public sigc::trackable
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param reports_area An observing reference to the view of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    ReportsAreaGenerateLaTeXPopover(Gtk::Builder &builder, const ReportsArea &reports_area);

private:
    /**
     * @brief Handle a click of the <i>Confirm</i> button by generating LaTeX with latexmk and providing
     * real-time feedback
     */
    void confirm_button_clicked();

    /**
     * @brief Handle a click of the <i>Cancel</i> button by disregarding any existing user input
     */
    void cancel_button_clicked();

    /**
     * @brief Resets all visible UI elements to their original state
     */
    void clear_inputs();

    /**
     * @brief Handle a click of the <i>Open Output Directory</i> button by temporarily hiding the popover and
     * raising a native Gtk::FileDialog file-chooser.
     * @details A callback is registered on the error-tolerant
     *  @ref open_directory_finished(const Glib::RefPtr<Gio::AsyncResult>&) to handle the completion of the
     * dialog. Until the dialog is closed/completed, the entire popover will remain hidden and inoperable.
     */
    void open_directory_button_clicked();

    /**
     * @brief Handle a completed session of the directory-chooser dialog and update any changes in the popover
     * state
     * @param result The asynchronously produced result of the Gtk::FileDialog session
     */
    void open_directory_finished(const Glib::RefPtr<Gio::AsyncResult> &result);

    /**
     * @brief Handle completion of the asynchronous executor.
     * @note This callback does not reset the generator instance, as it would be typically called from the
     * instance. Hence, the generator instance persists in memory until it is replaced by another despatcher.
     */
    void post_generation_callback() const;

    /**
     * @brief Handle a toggle of the <i>Show Details</i> button by showing or hiding the latexmk/pdflatex
     * output
     */
    void show_details_toggled() const;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    const ReportsArea &reports_area;
    const Glib::RefPtr<Gtk::TextBuffer> buffer;
    Gtk::Popover *const my_popover;
    Gtk::Box *const details_container;
    Gtk::Entry *const output_directory_entry;
    Gtk::Button *const confirm_button;
    Gtk::Button *const cancel_button;
    const Glib::RefPtr<Gtk::FileDialog> open_dialog;
    Gtk::CheckButton *const show_details_check;

    Glib::RefPtr<Gio::File> output_directory;
    std::optional<LaTeXReportGenerator> generator;
};

} // namespace optifol
```

```
#endif // REPORTSAREAGENERATELATEXPOPOVER_HPP
```

1.8.16.4 ReportsArea.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Main Window's Releases and Reports area
 * @author Oliver Dixon
 * @date 2025-07-08
 * @version Development
 */

#ifndef REPORTSAREA_HPP
#define REPORTSAREA_HPP

#include <gtkmm/builder.h>
#include <gtkmm/columnview.h>

#include "../../Storage/Subsystem.hpp"
#include "../ContextButtonCorrespondence.hpp"
#include "../IWindowArea.hpp"
#include "ReportsAreaGenerateLaTeXPopover.hpp"

namespace optifol
{

/**
 * @class ReportsArea
 * @brief Manage the <i>Releases and Reports</i> area
 *
 * @details
 *  <p>
 *      The <i>Releases and Reports</i> area provides controls for reviewing baselined releases of the project
 * with a frozen set of Requirement objects, and also enables the automatic generation of reports to document
 * baselines. The following GTK elements are expected from the given Gtk::Builder: <table> <tr> <th>GTK C++
 * Class</th> <th>Unique Identifier</th> <th>Purpose</th>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Widget (abstract)</td>
 *              <td><code>reports_advice_unselected</code></td>
 *              <td>Advice to display when the area is unavailable</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Widget (abstract)</td>
 *              <td><code>reports_content</code></td>
 *              <td>Replacement to <code>reports_advice_unselected</code>, containing all active content</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::ColumnView</td>
 *              <td><code>reports_elements_view</code></td>
 *              <td>Table containing baselines and previously generated reports</td>
 *          </tr>
 *          <tr>
 *              <td>Gio::Menu</td>
 *              <td><code>reports_context_menu</code></td>
 *              <td>Area-wide context menu</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::MenuButton</td>
 *              <td><code>reports_generate_latex</code></td>
 *              <td>Button for the LaTeX generation popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Popover</td>
 *              <td><code>reports_generate_latex_popover</code></td>
 *              <td>Popover for the LaTeX generation facility</td>
 *          </tr>
 *      </table>
 *      A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in
 * the expected type instantiations.
 *  </p>
 *  <p>
 *      In addition to the stated required GTK elements, constituent popovers of this view will require their
 * own, possibly distinct, set of elements: <ul> <li>@ref ReportsAreaGenerateLaTeXPopover</li>
```

```
 *      </ul>
 *  </p>
 */
class ReportsArea : public IWindowArea
{
public:
    explicit ReportsArea(Gtk::Builder &builder);

    void select_model(const Glib::RefPtr<Subsystem> &subsystem) override;

    void deselect_model() override;

    Subsystem *get_active_subsystem() noexcept override;

    const Subsystem *observe_active_subsystem() const noexcept override;

private:
    static const char *const area_name;

    std::pair<Gtk::Widget *, Gtk::Widget *> on_off_widgets;

    Gtk::ColumnView *view;

    ContextButtonCorrespondence context_menu;

    Glib::RefPtr<Subsystem> active_subsystem;

    ReportsAreaGenerateLaTeXPopover generate_latex_popover;
};

} // namespace optifol

#endif // REPORTSAREA_HPP
```

1.8.17 RequirementsIndexArea

1.8.17.1 IndexDeleteRequirementPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Delete Requirement UI popover
 * @author Oliver Dixon
 * @date 2025-07-10
 * @version Development
 */

#include "IndexDeleteRequirementPopover.hpp"
#include "../../Storage/Subsystem.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "RequirementsIndexArea.hpp"

namespace optifol
{

const char *const IndexDeleteRequirementPopover::popover_name = "Delete Requirement Popover";
const log4cxx::LoggerPtr IndexDeleteRequirementPopover::popover_logger =
        Logging::get_logger({"GUI", "RequirementsIndex", "DeleteRequirement"});

IndexDeleteRequirementPopover::IndexDeleteRequirementPopover(
        Gtk::Builder &builder, RequirementsIndexArea &index_area) :
    index_area(index_area),
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "delete_requirement_popover")),
    confirm_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "delete_requirement_confirm")),
    cancel_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "delete_requirement_cancel")),
    name_entry(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "delete_requirement_property_name"))
{
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &IndexDeleteRequirementPopover::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &IndexDeleteRequirementPopover::cancel_button_clicked));
    my_popover->signal_show().connect(sigc::mem_fun(*this, &IndexDeleteRequirementPopover::show_popover));
}
```

```
void IndexDeleteRequirementPopover::confirm_button_clicked() const
{
    my_popover->popdown();
    const auto slated_requirement = index_area.get_selection();
    index_area.get_active_subsystem()->delete_object(slated_requirement);
    popover_logger->info("Deleted a subsystem requirement with name \"" +
            slated_requirement->property_name().get_value() + "\".");
}

void IndexDeleteRequirementPopover::cancel_button_clicked() const
{
    my_popover->popdown();
    clear_inputs();
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from button-click callback.
void IndexDeleteRequirementPopover::clear_inputs() const
{
    name_entry->set_text("");
}
void IndexDeleteRequirementPopover::show_popover() const
{
    name_entry->set_text(index_area.get_selection()->property_name().get_value());
}

} // namespace optifol
```

1.8.17.2 IndexDeleteRequirementPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Delete Requirement UI popover
 * @author Oliver Dixon
 * @date 2025-07-10
 * @version Development
 */

#ifndef INDEXDELETEREQUIREMENTPOPOVER_HPP
#define INDEXDELETEREQUIREMENTPOPOVER_HPP

#include <gtkmm/builder.h>
#include <gtkmm/button.h>
#include <gtkmm/entry.h>
#include <gtkmm/popover.h>
#include <log4cxx/logger.h>

namespace optifol
{

class RequirementsIndexArea;

/**
 * @class IndexDeleteRequirementPopover
 * @brief Manage the <i>Delete Requirement</i> popover for the Requirements Index area
 * @see RequirementsIndexArea
 *
 * @details
 *  The <i>Delete Requirements</i> popover provides controls for deleting an existing Requirement from the
 * requirements index for the currently selected Subsystem. The following GTK elements are expected to be
 * available from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th> <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>delete_requirement_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>delete_requirement_confirm</code></td>
 *          <td>Confirm deletion of the named Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
```

```
 *          <td><code>delete_requirement_cancel</code></td>
 *          <td>Cancels deletion of a Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>delete_requirement_property_name</code></td>
 *          <td>Text entry area for the name of the new Requirement to delete</td>
 *      </tr>
 *  </table>
 */
class IndexDeleteRequirementPopover : public sigc::trackable
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param index_area A mutating reference to the view of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    IndexDeleteRequirementPopover(Gtk::Builder &builder, RequirementsIndexArea &index_area);

private:
    /**
     * @brief Handle a click of the <i>Confirm</i> by attempting to delete the named Requirement.
     */
    void confirm_button_clicked() const;

    /**
     * @brief Handle a click of the <i>Cancel</i> button by discarding all input and closing the popover.
     */
    void cancel_button_clicked() const;

    /**
     * @brief Clear all user fields in the popover
     */
    void clear_inputs() const;

    /**
     * @brief Populate all fields given the parental area's selection context and show the popover.
     */
    void show_popover() const;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    RequirementsIndexArea &index_area;
    Gtk::Popover *const my_popover;
    Gtk::Button *const confirm_button;
    Gtk::Button *const cancel_button;
    Gtk::Entry *const name_entry;
};

} // namespace optifol

#endif // INDEXDELETEREQUIREMENTPOPOVER_HPP
```

1.8.17.3 IndexDuplicateRequirementPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Duplicate Requirement UI popover
 * @author Oliver Dixon
 * @date 2025-07-10
 * @version Development
 */

#include "IndexDuplicateRequirementPopover.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "../Storage/Subsystem.hpp"
#include "RequirementsIndexArea.hpp"

namespace optifol
{
```

```cpp
const char *const IndexDuplicateRequirementPopover::popover_name = "Duplicate Requirement Popover";
const log4cxx::LoggerPtr IndexDuplicateRequirementPopover::popover_logger =
        Logging::get_logger({"GUI", "RequirementsIndex", "DuplicateRequirement"});

IndexDuplicateRequirementPopover::IndexDuplicateRequirementPopover(
        Gtk::Builder &builder, RequirementsIndexArea &index_area) :
    index_area(index_area),
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "duplicate_requirement_popover")),
    confirm_button(
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "duplicate_requirement_confirm")),
    cancel_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "duplicate_requirement_cancel")),
    old_name_entry(GTKHelpers::get_widget<Gtk::Entry>(
            popover_name, builder, "duplicate_requirement_property_old_name")),
    new_name_entry(GTKHelpers::get_widget<Gtk::Entry>(
            popover_name, builder, "duplicate_requirement_property_new_name"))
{
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &IndexDuplicateRequirementPopover::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &IndexDuplicateRequirementPopover::cancel_button_clicked));
    my_popover->signal_show().connect(sigc::mem_fun(*this, &IndexDuplicateRequirementPopover::show_popover));
}

void IndexDuplicateRequirementPopover::confirm_button_clicked() const noexcept
{
    my_popover->popdown();

    try {
        const auto subsystem = index_area.get_active_subsystem();
        subsystem->duplicate_requirement(*index_area.get_selection());

        popover_logger->info("Duplicated new Subsystem Requirement with name \"" +
                new_name_entry->get_text() + "\" from existing \"" + old_name_entry->get_text() + "\".");
    } catch (const std::runtime_error &error) {
        popover_logger->error("Could not duplicate Subsystem Requirement with name \"" +
                new_name_entry->get_text() + "\".");
        popover_logger->error(error.what());
    }

    clear_inputs();
}

void IndexDuplicateRequirementPopover::cancel_button_clicked() const
{
    my_popover->popdown();
    clear_inputs();
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from button-click callback.
void IndexDuplicateRequirementPopover::clear_inputs() const
{
    old_name_entry->set_text("");
    new_name_entry->set_text("");
}

void IndexDuplicateRequirementPopover::show_popover() const
{
    old_name_entry->set_text(index_area.get_selection()->property_name().get_value());
}

} // namespace optifol
```

1.8.17.4 IndexDuplicateRequirementPopover.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Duplicate Requirement UI popover
 * @author Oliver Dixon
 * @date 2025-07-10
 * @version Development
 */

#ifndef INDEXDUPLICATEREQUIREMENTPOPOVER_HPP
```

```
#define INDEXDUPLICATEREQUIREMENTPOPOVER_HPP

#include <gtkmm/builder.h>
#include <gtkmm/button.h>
#include <gtkmm/entry.h>
#include <gtkmm/popover.h>
#include <log4cxx/logger.h>

namespace optifol
{

class RequirementsIndexArea;

/**
 * @class IndexDuplicateRequirementPopover
 * @brief Manage the <i>Duplicate Requirement</i> popover for the Requirements Index area
 * @see RequirementsIndexArea
 *
 * @details
 *  The <i>Duplicate Requirements</i> popover provides controls for duplicating an existing Requirement from
 * the requirements index for the currently selected Subsystem. By default, the duplicated Requirement copies
 * all metadata from the existing Requirement, but can be edited separately. The following GTK elements are
 * expected to be available from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th> <th>Unique
 * Identifier</th> <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>duplicate_requirement_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>duplicate_requirement_confirm</code></td>
 *          <td>Confirm duplication of the named Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>duplicate_requirement_cancel</code></td>
 *          <td>Cancels duplication of a Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>duplicate_requirement_property_old_name</code></td>
 *          <td>Read-only text entry area for the name of the existing Requirement to duplicate</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>duplicate_requirement_property_new_name</code></td>
 *          <td>Text entry area for the name of the new Requirement</td>
 *      </tr>
 *  </table>
 */

class IndexDuplicateRequirementPopover : public sigc::trackable
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param index_area A mutating reference to the view of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    IndexDuplicateRequirementPopover(Gtk::Builder &builder, RequirementsIndexArea &index_area);

private:
    /**
     * @brief Handle a click of the <i>Confirm</i> by attempting to duplicate the named Requirement.
     */
    void confirm_button_clicked() const noexcept;

    /**
     * @brief Handle a click of the <i>Cancel</i> button by discarding all input and closing the popover.
     */
    void cancel_button_clicked() const;

    /**
     * @brief Clear all user fields in the popover
     */
    void clear_inputs() const;

    /**
```

```
     * @brief Populate all fields given the parental area's selection context and show the popover.
     */
    void show_popover() const;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    RequirementsIndexArea &index_area;
    Gtk::Popover *const my_popover;
    Gtk::Button *const confirm_button;
    Gtk::Button *const cancel_button;
    Gtk::Entry *const old_name_entry;
    Gtk::Entry *const new_name_entry;
};

} // namespace optifol

#endif // INDEXDUPLICATEREQUIREMENTPOPOVER_HPP
```

1.8.17.5 IndexNewRequirementPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the New Requirement UI popover
 * @author Oliver Dixon
 * @date 2025-07-10
 * @version Development
 */

#include "IndexNewRequirementPopover.hpp"
#include "../../Storage/Subsystem.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "RequirementsIndexArea.hpp"

namespace optifol
{

const char *const IndexNewRequirementPopover::popover_name = "New Requirement Popover";
const log4cxx::LoggerPtr IndexNewRequirementPopover::popover_logger =
        Logging::get_logger({"GUI", "RequirementsIndex", "NewRequirement"});

IndexNewRequirementPopover::IndexNewRequirementPopover(
        Gtk::Builder &builder, RequirementsIndexArea &index_area) :
    index_area(index_area),
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "new_requirement_popover")),
    confirm_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "new_requirement_confirm")),
    name_entry(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "new_requirement_property_name")),
    description_entry(GTKHelpers::get_widget<Gtk::TextView>(
            popover_name, builder, "new_requirement_property_description")),
    statement_entry(
            GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "new_requirement_property_sentence")),
    test_summary(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "new_requirement_test_count")),
    priority_entry(GTKHelpers::get_widget<Gtk::DropDown>(
            popover_name, builder, "new_requirement_property_priority")),
    manage_tests_popover(builder)
{
    // Get extra elements needed only for the constructor lifetime.
    const auto cancel_button =
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "new_requirement_cancel");

    // Set up the test management popover.
    const auto test_button =
            GTKHelpers::get_widget<Gtk::MenuButton>(popover_name, builder, "new_requirement_manage_tests");
    const auto test_manager =
            GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "manage_tests_popover");
    test_button->set_popover(*test_manager);

    // Set up buttons and self.
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &IndexNewRequirementPopover::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &IndexNewRequirementPopover::cancel_button_clicked));
    my_popover->signal_show().connect(sigc::mem_fun(*this, &IndexNewRequirementPopover::popover_shown));
```

```
    name_entry->signal_changed().connect(
            sigc::mem_fun(*this, &IndexNewRequirementPopover::name_entry_changed));

    // Force the popover into a known baseline state.
    clear_inputs();
}

void IndexNewRequirementPopover::confirm_button_clicked() noexcept
{
    my_popover->popdown();

    try {
        index_area.get_active_subsystem()->build_requirement(name_entry->get_text(),
                statement_entry->get_text(), description_entry->get_buffer()->get_text(),
                priority_entry->get_selected(), std::move(test_specification));

        popover_logger->info(
                "Created new Subsystem Requirement with name \"" + name_entry->get_text() + "\".");
    } catch (const std::runtime_error &error) {
        popover_logger->error(
                "Could not create Subsystem Requirement with name \"" + name_entry->get_text() + "\".");
        popover_logger->error(error.what());
    }
}

void IndexNewRequirementPopover::cancel_button_clicked() const
{
    my_popover->popdown();
}

void IndexNewRequirementPopover::popover_shown() noexcept
{
    clear_inputs(); // Clear down the inputs for the new entry.
    manage_tests_popover.set_model(test_specification);
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from button-click callback.
void IndexNewRequirementPopover::clear_inputs()
{
    name_entry->set_text("");
    description_entry->get_buffer()->set_text("");
    statement_entry->set_text("");
    priority_entry->set_selected(0);
    test_summary->set_text("");
    test_specification = Gio::ListStore<TestSpecificationEntry>::create();
    confirm_button->set_sensitive(false);

    test_specification->signal_items_changed().connect([this](guint, guint, guint) noexcept
            { test_summary->set_text(ManageTestsPopover::format_test_summary(*test_specification)); });
}

void IndexNewRequirementPopover::name_entry_changed() const noexcept
{
    confirm_button->set_sensitive(!name_entry->get_text().empty());
}

} // namespace optifol
```

1.8.17.6 IndexNewRequirementPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the New Requirement UI popover
 * @author Oliver Dixon
 * @date 2025-07-10
 * @version Development
 */

#ifndef INDEXNEWREQUIREMENTPOPOVER_HPP
#define INDEXNEWREQUIREMENTPOPOVER_HPP

#include <gtkmm/builder.h>
#include <gtkmm/dropdown.h>
#include <gtkmm/entry.h>
```

```
#include <gtkmm/textview.h>

#include "ManageTestsPopover.hpp"

namespace optifol
{

class RequirementsIndexArea;

/**
 * @class IndexNewRequirementPopover
 * @brief Manage the <i>New Requirement</i> popover for the Requirements Index area
 * @see RequirementsIndexArea for the parent area
 *
 * @details
 *  The <i>New Requirement</i> popover provides controls for creating a new Requirement for the currently
 * selected Subsystem. The following GTK elements are expected to be available from the given Gtk::Builder:
 *  <table>
 *      <tr>
 *          <th>GTK C++ Class</th>
 *          <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>new_requirement_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>new_requirement_confirm</code></td>
 *          <td>Confirm creation of a new Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>new_requirement_cancel</code></td>
 *          <td>Cancels creation of a new Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>new_requirement_property_name</code></td>
 *          <td>Text entry area for the name of the new Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::TextView</td>
 *          <td><code>new_requirement_property_description</code></td>
 *          <td>Multi-line text entry area for the description of the new Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>new_requirement_property_sentence</code></td>
 *          <td>Text entry area for the FOL statement of the new Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>new_requirement_test_count</code></td>
 *          <td>Read-only area for the test summary of the new Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::DropDown</td>
 *          <td><code>new_requirement_property_priority</code></td>
 *          <td>Text entry area for the optional associated test(s) of the new Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>manage_tests_popover</code></td>
 *          <td>Popover for managing tests</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>new_requirement_manage_tests</code></td>
 *          <td>Button for activating <code>manage_tests_popover</code> popover</td>
 *      </tr>
 *  </table>
 *  A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in the
 * expected type instantiations.
 */
class IndexNewRequirementPopover : public sigc::trackable
{
public:
    /**
```

```
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param index_area A mutating reference to the view of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    IndexNewRequirementPopover(Gtk::Builder &builder, RequirementsIndexArea &index_area);

private:
    /**
     * @brief Handle a click of the <i>Confirm</i> by attempting to create a Requirement with the given
     * characteristics.
     */
    void confirm_button_clicked() noexcept;

    /**
     * @brief Handle a click of the <i>Cancel</i> button by discarding all input and closing the popover.
     */
    void cancel_button_clicked() const;

    /**
     * @brief Handle a show of the popover by selecting the correct test specification model for the
     *  @ref manage_tests_popover.
     */
    void popover_shown() noexcept;

    /**
     * @brief Clear all user fields in the popover.
     */
    void clear_inputs();

    /**
     * @brief Handle a new committed Requirement name by enabling the <i>Confirm</i> button.
     */
    void name_entry_changed() const noexcept;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    RequirementsIndexArea &index_area;

    Gtk::Popover *const my_popover;
    Gtk::Button *const confirm_button;
    Gtk::Entry *const name_entry;
    Gtk::TextView *const description_entry;
    Gtk::Entry *const statement_entry;
    Gtk::Entry *const test_summary;
    Gtk::DropDown *const priority_entry;

    ManageTestsPopover manage_tests_popover;
    Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> test_specification;
};

} // namespace optifol

#endif // INDEXNEWREQUIREMENTPOPOVER_HPP
```

1.8.17.7 ManageTestsPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the ManageTestsPopover GUI popover
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#include "ManageTestsPopover.hpp"
#include "../../UserTesting/Discovery/DiscoveryTestFixture.hpp"
#include "../../UserTesting/Discovery/GoogleTestDiscoveryExecutable.hpp"
#include "../../UserTesting/Modelling/Test.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "RequirementsIndexArea.hpp"
```

```
namespace optifol
{

const char *const ManageTestsPopover::popover_name = "Manage Tests Popover";
const log4cxx::LoggerPtr ManageTestsPopover::popover_logger =
        Logging::get_logger({"GUI", "RequirementsIndex", "NewRequirement", "TestManagement"});

ManageTestsPopover::ManageTestsPopover(Gtk::Builder &builder) :
    popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "manage_tests_popover")),
    confirm_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "manage_tests_confirm")),
    new_test_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "manage_tests_new_test")),
    duplicate_test_button(
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "manage_tests_duplicate_test")),
    delete_test_button(
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "manage_tests_delete_test")),
    view(GTKHelpers::get_widget<Gtk::ColumnView>(popover_name, builder, "manage_tests_view"))
{
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &ManageTestsPopover::confirm_button_clicked));
    new_test_button->signal_clicked().connect(sigc::mem_fun(*this, &ManageTestsPopover::new_test_clicked));
    duplicate_test_button->signal_clicked().connect(
            sigc::mem_fun(*this, &ManageTestsPopover::duplicate_test_clicked));
    delete_test_button->signal_clicked().connect(
            sigc::mem_fun(*this, &ManageTestsPopover::delete_test_clicked));

    selection_model->set_autoselect(false);
    selection_model->set_can_unselect(true);

    view->set_model(selection_model);

    const auto columns = view->get_columns();
    const auto column_count = columns->get_n_items();

    for (guint position = 0; position < column_count; ++position) {
        Glib::RefPtr<Gtk::ColumnViewColumn> column = nullptr;

        if ((column = columns->get_typed_object<Gtk::ColumnViewColumn>(position)) != nullptr) {
            const auto &gtk_id = column->get_id();
            const auto factory = Gtk::SignalListItemFactory::create();

            if (gtk_id == "manage_tests_target_exe") {
                factory->signal_setup().connect(sigc::ptr_fun(&GTKHelpers::setup_entry));
                factory->signal_bind().connect(
                        sigc::mem_fun(*this, &ManageTestsPopover::bind_test_executable));
            } else if (gtk_id == "manage_tests_fixture") {
                factory->signal_setup().connect(sigc::ptr_fun(&ManageTestsPopover::setup_fixtures_combo));
                factory->signal_bind().connect(sigc::ptr_fun(&ManageTestsPopover::bind_test_fixture));
            } else if (gtk_id == "manage_tests_name") {
                factory->signal_setup().connect(sigc::ptr_fun(&GTKHelpers::setup_combo_box));
                factory->signal_bind().connect(sigc::ptr_fun(&ManageTestsPopover::bind_test_name));
            } else
                // Jump out here if unrecognised, so all further code can assume a factory was configured.
                continue;

            column->set_factory(factory);
        }
    }
}

void ManageTestsPopover::set_model(const Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &model) noexcept
{
    test_spec_model = model;
    selection_model->set_model(test_spec_model);
}

std::string ManageTestsPopover::format_test_summary(const Gio::ListModel &test_model) noexcept
{
    const auto test_count = test_model.get_n_items();

    if (test_count == 0)
        return "";

    if (test_count == 1)
        return "1 test defined";

    return std::to_string(test_count) + " tests defined";
}

void ManageTestsPopover::confirm_button_clicked() const
{
    popover->popdown();
```

```
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive. Invoked from callbacks.
void ManageTestsPopover::new_test_clicked() const
{
    test_spec_model->append(
            Glib::make_refptr_for_instance<TestSpecificationEntry>(new TestSpecificationEntry()));
}

void ManageTestsPopover::delete_test_clicked() const
{
    test_spec_model->remove(selection_model->get_selected());
}

void ManageTestsPopover::duplicate_test_clicked() const
{
    const auto untyped_selection = selection_model->get_selected_item();

    if (untyped_selection == nullptr) {
        new_test_clicked();
        return;
    }

    const auto typed_selection = dynamic_cast<const TestSpecificationEntry *>(untyped_selection.get());

    if (typed_selection == nullptr)
        new_test_clicked();
    else
        test_spec_model->append(Glib::make_refptr_for_instance(new TestSpecificationEntry(*typed_selection)));
}

void ManageTestsPopover::setup_fixtures_combo(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto combo_box = Gtk::make_managed<Gtk::DropDown>();
    combo_box->set_halign(Gtk::Align::START);
    list_item->set_child(*combo_box);

    const auto factory = Gtk::SignalListItemFactory::create();

    factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
    factory->signal_bind().connect(
            [](const Glib::RefPtr<Gtk::ListItem> &selected_item)
            {
                const auto target_label = dynamic_cast<Gtk::Label *>(selected_item->get_child());
                const auto typed_fixture =
                        std::dynamic_pointer_cast<DiscoveryTestFixture>(selected_item->get_item());
                if (target_label == nullptr || typed_fixture == nullptr)
                    return;

                target_label->set_text(typed_fixture->property_name().get_value());
            });

    combo_box->set_factory(factory);
}

void ManageTestsPopover::bind_test_executable(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto exe_entry = dynamic_cast<Gtk::Entry *>(list_item->get_child());
    const auto test_spec = std::dynamic_pointer_cast<TestSpecificationEntry>(list_item->get_item());

    if (exe_entry == nullptr || test_spec == nullptr)
        return;

    // Set-up the binding from the TestSpecificationEntry object to the GUI, such that model changes are
    // visible.
    Glib::Binding::bind_property(test_spec->property_executable(), exe_entry->property_text(),
            Glib::Binding::Flags::SYNC_CREATE,
            [](const Glib::RefPtr<DiscoveryTestExecutable> &exe) -> std::optional<Glib::ustring>
            {
                if (exe == nullptr)
                    return "";

                return exe->property_name().get_value();
            });

    // Set-up the binding from the view to the model, such that committed changes are propagated and handled.
    exe_entry->signal_activate().connect(sigc::bind(
            sigc::mem_fun(*this, &ManageTestsPopover::handle_executable_change), test_spec, exe_entry));
}

void ManageTestsPopover::bind_test_fixture(const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept
```

```
{
    const auto fixture_combo = dynamic_cast<Gtk::DropDown *>(list_item->get_child());
    const auto test_spec = std::dynamic_pointer_cast<TestSpecificationEntry>(list_item->get_item());

    if (fixture_combo == nullptr || test_spec == nullptr)
        return;

    /*
     * Set-up the binding from the TestSpecificationEntry object to the GUI, such that changes to the
     * executable will propagate to the fixtures combo box by loading the correct fixture model. This needn't
     * be a bidirectional binding as the combo box model would only ever change due to a change in the
     * TestSpecificationEntry.
     */
    Glib::Binding::bind_property(test_spec->property_executable(), fixture_combo->property_model(),
            Glib::Binding::Flags::SYNC_CREATE,
            [](const Glib::RefPtr<DiscoveryTestExecutable> &exe) noexcept
                    -> std::optional<Glib::RefPtr<Gio::ListModel>>
            {
                if (exe == nullptr)
                    return {};

                return exe->get_fixture_model();
            });

    // Set-up the binding from the model to the view, such that committed changes propagate to the fixture
    // combo box.
    Glib::Binding::bind_property(fixture_combo->property_selected_item(), test_spec->property_fixture(),
            Glib::Binding::Flags::SYNC_CREATE,
            [](const Glib::RefPtr<Glib::ObjectBase> &selected_item) noexcept
                    -> std::optional<Glib::RefPtr<DiscoveryTestFixture>>
            {
                const auto &typed_entry = std::dynamic_pointer_cast<DiscoveryTestFixture>(selected_item);
                if (typed_entry == nullptr)
                    return {};

                return typed_entry;
            });
}

void ManageTestsPopover::bind_test_name(const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept
{
    const auto test_combo = dynamic_cast<Gtk::DropDown *>(list_item->get_child());
    const auto test_spec = std::dynamic_pointer_cast<TestSpecificationEntry>(list_item->get_item());

    Glib::Binding::bind_property(test_spec->property_fixture(), test_combo->property_model(),
            Glib::Binding::Flags::SYNC_CREATE,
            [](const Glib::RefPtr<DiscoveryTestFixture> &fixture) noexcept
                    -> std::optional<Glib::RefPtr<Gtk::StringList>>
            {
                if (fixture == nullptr)
                    return {};

                return fixture->get_test_model();
            });

    Glib::Binding::bind_property(test_combo->property_selected_item(), test_spec->property_name(),
            Glib::Binding::Flags::SYNC_CREATE,
            [](const Glib::RefPtr<Glib::ObjectBase> &selected_item) noexcept -> std::optional<Glib::ustring>
            {
                const auto &typed_entry = dynamic_cast<Gtk::StringObject *>(selected_item.get());
                if (typed_entry == nullptr)
                    return {};

                return typed_entry->get_string();
            });
}

void ManageTestsPopover::handle_executable_change(
        const Glib::RefPtr<TestSpecificationEntry> &test_spec, const Gtk::Entry *const exe_entry)
{
    const auto cached_exe_it = discovery_exe_cache.find(exe_entry->get_text());

    if (cached_exe_it == discovery_exe_cache.cend()) {
        const auto &name = exe_entry->get_text();
        const auto [entry, was_added] = discovery_exe_cache.emplace(
                Glib::make_refptr_for_instance(new GoogleTestDiscoveryExecutable(name)));

        if (was_added) {
            popover_logger->debug("Created and cached new test executable for discovery: \"" + name + "\".");
            test_spec->property_executable().set_value(*entry);
        } else
```

```
            popover_logger->error("Could not cache new test executable for discovery: \"" + name + "\".");
    } else {
        test_spec->property_executable().set_value(*cached_exe_it);
        popover_logger->debug("Recovered test executable for discovery from the cache: \"" +
                (*cached_exe_it)->property_name().get_value() + "\".");
    }
}

} // namespace optifol
```

1.8.17.8 ManageTestsPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the ManageTestsPopover GUI popover
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#ifndef MANAGETESTSPOPOVER_HPP
#define MANAGETESTSPOPOVER_HPP

#include <giomm/liststore.h>
#include <gtkmm/button.h>
#include <gtkmm/columnview.h>
#include <gtkmm/entry.h>
#include <gtkmm/listitem.h>
#include <gtkmm/popover.h>
#include <gtkmm/singleselection.h>
#include <log4cxx/logger.h>
#include <unordered_set>

#include "../../Optifol.hpp"
#include "../../UserTesting/Discovery/DiscoveryTestExecutable.hpp"

namespace optifol
{

class Test;
class RequirementsIndexArea;
class TestSpecificationEntry;

/**
 * @class ManageTestsPopover
 * @brief Manage the test-management capability in the Requirement editor.
 * @see IndexNewRequirementPopover for a possible parent area
 * @see IndexEditRequirementPopover for a possible parent area
 *
 * @details
 *  The <i>Test Management</i> popover is a sub-popover of the IndexNewRequirementPopover or the
 *  IndexEditRequirementPopover. It provides controls to review and intelligently define software-level unit
 * tests for a fixed Requirement i.a.w. a test executable. For the purposes of querying test executables, the
 * popover also maintains a cache of Subsystem-agnostic discovery executables. The following GTK elements are
 * expected from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th> <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>manage_tests_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>manage_tests_confirm</code></td>
 *          <td>Confirm test specifications</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>manage_tests_new_test</code></td>
 *          <td>Create a new TestSpecificationEntry</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
```

```
 *          <td><code>manage_tests_duplicate_test</code></td>
 *          <td>Duplicate the selected TestSpecificationEntry</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>manage_tests_delete_test</code></td>
 *          <td>Delete the selected TestSpecificationEntry</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnView</td>
 *          <td><code>manage_tests_view</code></td>
 *          <td>View for the dynamically populated TestSpecificationEntry list</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>manage_tests_target_exe</code></td>
 *          <td>View column for the target executable name Gtk::Entry</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>manage_tests_fixture</code></td>
 *          <td>View column for the target test fixture Gtk::DropDown</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>manage_tests_name</code></td>
 *          <td>View column for the target test case name Gtk::DropDown</td>
 *      </tr>
 *  </table>
 *  A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in the
 * expected type instantiations.
 */
class ManageTestsPopover : public sigc::trackable
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    explicit ManageTestsPopover(Gtk::Builder &builder);

    /**
     * @brief Sets a new model for the TestSpecificationEntry objects, discarding the existing one.
     * @param model The new model.
     */
    void set_model(const Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &model) noexcept;

    /**
     * @brief Provide a human-readable summary of a model containing tests or specifications thereof.
     * @param test_model The test model to query.
     * @see Test
     * @see TestSpecificationEntry
     */
    static std::string format_test_summary(const Gio::ListModel &test_model) noexcept;

private:
    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    /**
     * @brief Handle a click of the <i>Confirm</i> button by hiding the popover. (The model is queried later.)
     */
    void confirm_button_clicked() const;

    void new_test_clicked() const;

    void delete_test_clicked() const;

    void duplicate_test_clicked() const;

    static void setup_fixtures_combo(const Glib::RefPtr<Gtk::ListItem> &list_item);

    void bind_test_executable(const Glib::RefPtr<Gtk::ListItem> &list_item);

    static void bind_test_fixture(const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept;

    static void bind_test_name(const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept;

    void handle_executable_change(
        const Glib::RefPtr<TestSpecificationEntry> &test_spec, const Gtk::Entry *exe_entry);
```

```
    Gtk::Popover *const popover;
    Gtk::Button *const confirm_button;
    Gtk::Button *const new_test_button;
    Gtk::Button *const duplicate_test_button;
    Gtk::Button *const delete_test_button;
    Gtk::ColumnView *const view;

    Glib::RefPtr<Gtk::SingleSelection> selection_model = Gtk::SingleSelection::create();
    Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> test_spec_model;

    SharedUnorderedSet<DiscoveryTestExecutable> discovery_exe_cache;
};

} // namespace optifol

#endif // MANAGETESTSPOPOVER_HPP
```

1.8.17.9 RequirementsIndexArea.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Main Window's Requirements Index UI area
 * @author Oliver Dixon
 * @date 2025-03-29
 * @version Development
 */

#include "RequirementsIndexArea.hpp"
#include "../../Storage/Subsystem.hpp"
#include "../GTKHelpers.hpp"

namespace optifol
{

const char *const RequirementsIndexArea::area_name = "Requirements Index Area";

RequirementsIndexArea::RequirementsIndexArea(Gtk::Builder &builder) :
    on_off_widgets(
            GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "requirements_index_advice_unselected"),
            GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "requirements_index_content")),
    view(GTKHelpers::get_widget<Gtk::ColumnView>(area_name, builder, "requirements_view")),
    context_menu(view, GTKHelpers::get_object<Gio::Menu>(area_name, builder, "requirement_context_menu"),
            {{"new_requirement",
                    GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "new_requirement"),
                    GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "new_requirement_popover"),
                    true},
                    {"edit_requirement",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "edit_requirement"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "edit_requirement_popover"),
                            false},
                    {"delete_requirement",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "delete_requirement"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "delete_requirement_popover"),
                            false},
                    {"duplicate_requirement",
                            GTKHelpers::get_widget<Gtk::MenuButton>(
                                    area_name, builder, "duplicate_requirement"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "duplicate_requirement_popover"),
                            false}}),
    new_requirement_popover(builder, *this),
    edit_requirement_popover(builder, *this),
    duplicate_requirement_popover(builder, *this),
    delete_requirement_popover(builder, *this)
{
    selection_model->set_autoselect(false);
    selection_model->set_can_unselect(true);

    selection_model->signal_selection_changed().connect(
            [this](guint, const guint n_items)
            {
```

```
                if (n_items == 0) {
                    context_menu.disable_action("edit_requirement");
                    context_menu.disable_action("delete_requirement");
                    context_menu.disable_action("duplicate_requirement");
                } else {
                    context_menu.enable_action("edit_requirement");
                    context_menu.enable_action("delete_requirement");
                    context_menu.enable_action("duplicate_requirement");
                }
            });

    selection_model->signal_items_changed().connect(
            [this](guint, const guint removed, guint)
            {
                if (removed > 0) {
                    // If anything was removed from the model, just deselect everything out of an abundance of
                    // caution.
                    context_menu.disable_action("edit_requirement");
                    context_menu.disable_action("delete_requirement");
                    context_menu.disable_action("duplicate_requirement");
                }
            });

    view->set_model(selection_model);

    const auto columns = view->get_columns();
    const auto column_count = columns->get_n_items();

    for (guint position = 0; position < column_count; ++position) {
        Glib::RefPtr<Gtk::ColumnViewColumn> column = nullptr;

        if ((column = columns->get_typed_object<Gtk::ColumnViewColumn>(position)) != nullptr) {
            const auto &gtk_id = column->get_id();
            const auto factory = Gtk::SignalListItemFactory::create();

            if (gtk_id == "requirement_name") {
                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(sigc::ptr_fun(&StorageObjectBase::bind_name));
            } else if (gtk_id == "requirement_statement") {
                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, true));
                factory->signal_bind().connect(
                        sigc::ptr_fun(&RequirementsIndexArea::on_bind_property_statement));
            } else if (gtk_id == "requirement_description") {
                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(
                        sigc::ptr_fun(&RequirementsIndexArea::on_bind_property_description));
            } else if (gtk_id == "requirement_priority") {
                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(
                        sigc::ptr_fun(&RequirementsIndexArea::on_bind_property_priority));
            } else if (gtk_id == "requirement_test") {
                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(sigc::ptr_fun(&RequirementsIndexArea::on_bind_property_test));
            } else if (gtk_id == "requirement_created") {
                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(sigc::ptr_fun(&StorageObjectBase::bind_creation_time));
            } else if (gtk_id == "requirement_modified") {
                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(sigc::ptr_fun(&StorageObjectBase::bind_modification_time));
            } else
                // Jump out here if unrecognised, so all further code can assume a factory was configured.
                continue;

            column->set_factory(factory);
        }
    }
}

void RequirementsIndexArea::select_model(const Glib::RefPtr<Subsystem> &new_subsystem)
{
    on_off_widgets.first->set_visible(false);
    on_off_widgets.second->set_visible(true);

    active_subsystem = new_subsystem;
    new_subsystem->populate_selection_model(*selection_model);
}

void RequirementsIndexArea::deselect_model()
{
    on_off_widgets.second->set_visible(false);
    on_off_widgets.first->set_visible(true);
```

```
    active_subsystem = nullptr;
    selection_model->set_model(nullptr);
}

Subsystem *RequirementsIndexArea::get_active_subsystem() noexcept
{
    return active_subsystem.get();
}

const Subsystem *RequirementsIndexArea::observe_active_subsystem() const noexcept
{
    return active_subsystem.get();
}

Glib::RefPtr<Requirement> RequirementsIndexArea::get_selection() const
{
    const auto selected_item = selection_model->get_selected_item();

    if (selected_item == nullptr)
        throw std::runtime_error("No item selected in the selection model.");

    const auto selected_requirement = std::dynamic_pointer_cast<Requirement>(selected_item);

    if (selected_requirement == nullptr)
        throw std::runtime_error("Selected item is not a Requirement.");

    return selected_requirement;
}

void RequirementsIndexArea::on_bind_property_description(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
    const auto item = std::dynamic_pointer_cast<Requirement>(list_item->get_item());

    if (label != nullptr && item != nullptr)
        Glib::Binding::bind_property(
                item->property_description(), label->property_label(), Glib::Binding::Flags::SYNC_CREATE);
}

void RequirementsIndexArea::on_bind_property_statement(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
    const auto item = std::dynamic_pointer_cast<Requirement>(list_item->get_item());

    if (label != nullptr && item != nullptr)
        Glib::Binding::bind_property(item->property_statement(), label->property_label(),
                Glib::Binding::Flags::SYNC_CREATE,
                [item](const Glib::ustring &) { return item->get_formatted_statement(); });
}

void RequirementsIndexArea::on_bind_property_priority(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
    const auto item = std::dynamic_pointer_cast<Requirement>(list_item->get_item());

    if (label != nullptr && item != nullptr)
        Glib::Binding::bind_property(
                item->property_priority(), label->property_label(), Glib::Binding::Flags::SYNC_CREATE);
}

void RequirementsIndexArea::on_bind_property_test(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
    const auto item = std::dynamic_pointer_cast<Requirement>(list_item->get_item());

    if (label != nullptr && item != nullptr) {
        const auto typed_tests = item->get_tests();
        if (typed_tests != nullptr) {
            label->set_text(ManageTestsPopover::format_test_summary(*typed_tests));
            typed_tests->signal_items_changed().connect([label, typed_tests](guint, guint, guint) noexcept
                    { label->set_text(ManageTestsPopover::format_test_summary(*typed_tests)); });
        }
    }
}

void RequirementsIndexArea::on_bind_property_normalised(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
    const auto item = std::dynamic_pointer_cast<Requirement>(list_item->get_item());
```

```
    if (label != nullptr && item != nullptr)
        Glib::Binding::bind_property(
                item->property_normalised(), label->property_label(), Glib::Binding::Flags::SYNC_CREATE);
}

} // namespace optifol
```

1.8.17.10 RequirementsIndexArea.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Main Window's Requirements Index UI area
 * @author Oliver Dixon
 * @date 2025-03-29
 * @version Development
 */

#ifndef REQUIREMENTSINDEXAREA_HPP
#define REQUIREMENTSINDEXAREA_HPP

#include <giomm/liststore.h>
#include <gtkmm/builder.h>
#include <gtkmm/columnview.h>
#include <gtkmm/listitem.h>
#include <gtkmm/singleselection.h>

#include "../../Storage/Requirement.hpp"
#include "../ContextButtonCorrespondence.hpp"
#include "../IWindowArea.hpp"
#include "IndexDeleteRequirementPopover.hpp"
#include "IndexDuplicateRequirementPopover.hpp"
#include "IndexEditRequirementPopover.hpp"
#include "IndexNewRequirementPopover.hpp"

namespace optifol
{

/**
 * @class RequirementsIndexArea
 * @brief Manage the <i>Requirements Index</i> area
 * @details
 *  <p>
 *      The <i>Requirements Index</i> area provides controls for reviewing and manipulating requirements for a
 * single Subsystem. The basic set of operations on the atomic Requirement object includes creation, deletion,
 *      modification, and duplication. The following GTK elements are expected from the given Gtk::Builder:
 *      <table>
 *          <tr>
 *              <th>GTK C++ Class</th>
 *              <th>Unique Identifier</th>
 *              <th>Purpose</th>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Widget (abstract)</td>
 *              <td><code>requirements_index_advice_unselected</code></td>
 *              <td>Advice to display when the area is unavailable</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Widget (abstract)</td>
 *              <td><code>requirements_index_content</code></td>
 *              <td>Replacement to <code>requirements_index_advice_unselected</code>, containing all active
 * content</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::ColumnView</td>
 *              <td><code>requirements_view</code></td>
 *              <td>Table containing Requirement entries and associated metadata</td>
 *          </tr>
 *          <tr>
 *              <td>Gio::Menu</td>
 *              <td><code>requirement_context_menu</code></td>
 *              <td>Area-wide context menu</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::MenuButton</td>
```

```
 *              <td><code>new_requirement</code></td>
 *              <td>Button to active <i>New Requirement</i> popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Popover</td>
 *              <td><code>new_requirement_popover</code></td>
 *              <td>The <i>New Requirement</i> popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::MenuButton</td>
 *              <td><code>edit_requirement</code></td>
 *              <td>Button to active <i>Edit Requirement</i> popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Popover</td>
 *              <td><code>edit_requirement_popover</code></td>
 *              <td>The <i>Edit Requirement</i> popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::MenuButton</td>
 *              <td><code>duplicate_requirement</code></td>
 *              <td>Button to active <i>Duplicate Requirement</i> popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Popover</td>
 *              <td><code>duplicate_requirement_popover</code></td>
 *              <td>The <i>Duplicate Requirement</i> popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::MenuButton</td>
 *              <td><code>delete_requirement</code></td>
 *              <td>Button to active <i>Delete Requirement</i> popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::Popover</td>
 *              <td><code>delete_requirement_popover</code></td>
 *              <td>The <i>Delete Requirement</i> popover</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::ColumnViewColumn</td>
 *              <td><code>requirement_name</code></td>
 *              <td>The Requirement name column</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::ColumnViewColumn</td>
 *              <td><code>requirement_statement</code></td>
 *              <td>The Requirement FOL statement column</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::ColumnViewColumn</td>
 *              <td><code>requirement_description</code></td>
 *              <td>The Requirement multi-line description column</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::ColumnViewColumn</td>
 *              <td><code>requirement_priority</code></td>
 *              <td>The numerical Requirement priority column</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::ColumnViewColumn</td>
 *              <td><code>requirement_created</code></td>
 *              <td>The Requirement created-on date/time column</td>
 *          </tr>
 *          <tr>
 *              <td>Gtk::ColumnViewColumn</td>
 *              <td><code>requirement_modified</code></td>
 *              <td>The Requirement last-modified date/time column</td>
 *          </tr>
 *      </table>
 *  </p>
 *  <p>
 *      In addition to the stated required GTK elements, constituent popovers of this view will require their
 * own, possibly distinct, set of elements: <ul> <li>@ref IndexNewRequirementPopover</li> <li>@ref
 * IndexEditRequirementPopover</li> <li>@ref IndexDuplicateRequirementPopover</li> <li>@ref
 * IndexDeleteRequirementPopover</li>
 *      </ul>
 *  </p>
 */
class RequirementsIndexArea : public IWindowArea
{
public:
```

```
    /**
     * @brief Construct a new compartmentalised area for displaying and managing sets of subsystem
     * requirements
     * @param builder The GTK builder attached to the main window
     */
    explicit RequirementsIndexArea(Gtk::Builder &builder);

    void select_model(const Glib::RefPtr<Subsystem> &new_subsystem) override;

    void deselect_model() override;

    Subsystem *get_active_subsystem() noexcept override;

    const Subsystem *observe_active_subsystem() const noexcept override;

    Glib::RefPtr<Requirement> get_selection() const;

    /**
     * @brief Bind a Requirement description attribute to a label
     * @param list_item The container in which the destination label exists
     */
    static void on_bind_property_description(const Glib::RefPtr<Gtk::ListItem> &list_item);

    /**
     * @brief Bind a Requirement statement attribute to a label
     * @param list_item The container in which the destination label exists
     */
    static void on_bind_property_statement(const Glib::RefPtr<Gtk::ListItem> &list_item);

    /**
     * @brief Bind a Requirement priority attribute to a label
     * @param list_item The container in which the destination label exists
     */
    static void on_bind_property_priority(const Glib::RefPtr<Gtk::ListItem> &list_item);

    /**
     * @brief Bind a Requirement test attribute to a label
     * @param list_item The container in which the destination label exists
     */
    static void on_bind_property_test(const Glib::RefPtr<Gtk::ListItem> &list_item);

    /**
     * @brief Bind a Requirement CNF-normalised attribute to a label
     * @param list_item The container in which the destination label exists
     */
    static void on_bind_property_normalised(const Glib::RefPtr<Gtk::ListItem> &list_item);

private:
    Glib::RefPtr<Subsystem> active_subsystem;
    Glib::RefPtr<Gio::ListStore<Requirement>> data_model;
    Glib::RefPtr<Gtk::SingleSelection> selection_model = Gtk::SingleSelection::create();

    std::pair<Gtk::Widget *, Gtk::Widget *> on_off_widgets;

    static const char *const area_name;

    Gtk::ColumnView *view;

    ContextButtonCorrespondence context_menu;

    IndexNewRequirementPopover new_requirement_popover;
    IndexEditRequirementPopover edit_requirement_popover;
    IndexDuplicateRequirementPopover duplicate_requirement_popover;
    IndexDeleteRequirementPopover delete_requirement_popover;
};

} // namespace optifol

#endif
```

1.8.18 TestingArea

1.8.18.1 TestingArea.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */
```

```cpp
/**
 * @file
 * @brief Class implementation for the Main Window's Testing UI area
 * @author Oliver Dixon
 * @date 2025-07-09
 * @version Development
 */

#include "TestingArea.hpp"
#include "../../UserTesting/Modelling/TestGroup.hpp"
#include "../GTKHelpers.hpp"

namespace optifol
{

const log4cxx::LoggerPtr TestingArea::area_logger = Logging::get_logger({"GUI", "TestingCompliance"});
const char *const TestingArea::area_name = "Testing and Compliance Area";

TestingArea::TestingArea(Gtk::Builder &builder) :
    test_groups_view(GTKHelpers::get_widget<Gtk::ColumnView>(area_name, builder, "test_groups_view")),
    context_menu(test_groups_view,
            GTKHelpers::get_object<Gio::Menu>(area_name, builder, "test_groups_context_menu"),
            {{"new_test_group", GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "new_test_group"),
                    GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "new_test_group_popover"),
                    true},
                    {"rename_test_group",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "rename_test_group"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "rename_test_group_popover"),
                            true},
                    {"delete_test_group",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "delete_test_group"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "delete_test_group_popover"),
                            true},
                    {"copy_to_test_group",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "copy_to_test_group"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "copy_to_test_group_popover"),
                            true},
                    {"move_to_test_group",
                            GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "move_to_test_group"),
                            GTKHelpers::get_widget<Gtk::Popover>(
                                    area_name, builder, "move_to_test_group_popover"),
                            true},
                    {"run_tests", GTKHelpers::get_widget<Gtk::MenuButton>(area_name, builder, "run_tests"),
                            GTKHelpers::get_widget<Gtk::Popover>(area_name, builder, "run_tests_popover"),
                            true}}),
    on_off_widgets(GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "testing_advice_unselected"),
            GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "testing_content")),
    testing_failed_view(builder),
    new_test_group_popover(builder, *this),
    rename_test_group_popover(builder, *this),
    delete_test_group_popover(builder, *this),
    copy_requirement_popover(builder, *this),
    move_requirement_popover(builder, *this),
    run_tests_popover(builder, *this)
{
    configure_selection_model();
    test_groups_view->set_model(selection_model);

    configure_columns();
}

void TestingArea::select_model(const Glib::RefPtr<Subsystem> &new_subsystem)
{
    on_off_widgets.first->set_visible(false);
    on_off_widgets.second->set_visible(true);

    active_subsystem = new_subsystem;
    tree_model = Gtk::TreeListModel::create(
            active_subsystem->get_test_groups(), &ITestModelNode::get_given_tests_tree, true, true);
    selection_model->set_model(tree_model);
    testing_failed_view.select_model(new_subsystem);
}

void TestingArea::deselect_model()
{
    on_off_widgets.second->set_visible(false);
    on_off_widgets.first->set_visible(true);
```

```
    active_subsystem = nullptr;
    tree_model = nullptr;
    selection_model->set_model(nullptr);
    testing_failed_view.deselect_model();
}

Subsystem *TestingArea::get_active_subsystem() noexcept
{
    return active_subsystem.get();
}

const Subsystem *TestingArea::observe_active_subsystem() const noexcept
{
    return active_subsystem.get();
}

Glib::RefPtr<Gtk::TreeListRow> TestingArea::get_selected_row() noexcept
{
    return tree_model->get_row(selection_model->get_selected());
}

Glib::RefPtr<const Gtk::TreeListRow> TestingArea::get_selected_row() const noexcept
{
    return tree_model->get_row(selection_model->get_selected());
}

Glib::RefPtr<TestGroup> TestingArea::get_selected_test_group()
{
    auto selected_row = get_selected_row();
    if (selected_row == nullptr)
        throw std::runtime_error("Popover was made available despite no suitable Test Group selection.");

    while (selected_row->get_depth() > 0)
        selected_row = selected_row->get_parent();

    const auto test_group = std::dynamic_pointer_cast<TestGroup>(selected_row->get_item());
    if (test_group == nullptr)
        throw std::runtime_error("Popover could not find a suitable Test Group.");

    return test_group;
}

Glib::RefPtr<Requirement> TestingArea::get_selected_requirement()
{
    auto selected_row = get_selected_row();
    if (selected_row == nullptr || selected_row->get_depth() == 0)
        throw std::runtime_error("Popover was made available despite no suitable Requirement selection.");

    while (selected_row->get_depth() > 1)
        selected_row = selected_row->get_parent();

    const auto requirement = std::dynamic_pointer_cast<Requirement>(selected_row->get_item());
    if (requirement == nullptr)
        throw std::runtime_error("Popover could not find a suitable Requirement.");

    return requirement;
}

std::optional<Glib::ustring> TestingArea::bind_test_result(const std::shared_ptr<TestResult> &result) noexcept
{
    if (result == nullptr)
        return "Unknown";

    try {
        return std::string(result->has_passed() ? "Passed" : "Failed") + " in " +
                std::to_string(result->get_execution_time()) + " ms";
    } catch (...) {
        area_logger->error("Could not format Test Result string due to system error.");
        return "Unknown error in evaluation";
    }
}

void TestingArea::configure_columns() const
{
    const auto columns = test_groups_view->get_columns();
    const auto column_count = columns->get_n_items();

    for (guint position = 0; position < column_count; ++position) {
        Glib::RefPtr<Gtk::ColumnViewColumn> column = nullptr;

        if ((column = columns->get_typed_object<Gtk::ColumnViewColumn>(position)) != nullptr) {
```

```
            const auto &gtk_id = column->get_id();
            const auto factory = Gtk::SignalListItemFactory::create();

            if (gtk_id == "test_requirement_name") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_expandable_label, false));
                factory->signal_bind().connect([this](const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept
                        { StorageObjectBase::bind_name_property_expandable(list_item, tree_model); });

            } else if (gtk_id == "test_target_executable") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(
                        [](const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept -> void
                        {
                            const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
                            const auto typed_test = std::dynamic_pointer_cast<Test>(list_item->get_item());

                            if (label == nullptr || typed_test == nullptr)
                                return;

                            Glib::Binding::bind_property(typed_test->property_target_executable_name(),
                                    label->property_label(), Glib::Binding::Flags::SYNC_CREATE);
                        });

            } else if (gtk_id == "test_fixture") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(
                        [](const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept -> void
                        {
                            const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
                            const auto typed_test = std::dynamic_pointer_cast<Test>(list_item->get_item());

                            if (label == nullptr || typed_test == nullptr)
                                return;

                            Glib::Binding::bind_property(typed_test->property_fixture(),
                                    label->property_label(), Glib::Binding::Flags::SYNC_CREATE);
                        });

            } else if (gtk_id == "test_status") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(
                        [](const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept -> void
                        {
                            const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
                            const auto typed_test = std::dynamic_pointer_cast<Test>(list_item->get_item());

                            if (label == nullptr || typed_test == nullptr)
                                return;

                            Glib::Binding::bind_property(typed_test->property_result(),
                                    label->property_label(), Glib::Binding::Flags::SYNC_CREATE,
                                    sigc::ptr_fun(&TestingArea::bind_test_result));
                        });

            } else
                // Jump out here if unrecognised, so all further code can assume a factory was configured.
                continue;

            column->set_factory(factory);
        }
    }
}

void TestingArea::configure_selection_model() const
{
    selection_model->set_autoselect(false);
    selection_model->set_can_unselect(true);
}

} // namespace optifol
```

1.8.18.2 TestingArea.hpp

```
/*
 * Copyright (c) All Rights Reserved
```

```
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Main Window's Testing UI area
 * @author Oliver Dixon
 * @date 2025-07-09
 * @version Development
 */

#ifndef TESTINGAREA_HPP
#define TESTINGAREA_HPP

#include <glibmm/refptr.h>
#include <gtkmm/columnview.h>
#include <gtkmm/label.h>
#include <gtkmm/singleselection.h>
#include <gtkmm/treelistmodel.h>

#include "../../Storage/Subsystem.hpp"
#include "../../UserTesting/Execution/PayloadManagement/GoogleTestListener.hpp"
#include "../ContextButtonCorrespondence.hpp"
#include "../IWindowArea.hpp"
#include "TestingCopyToTestGroupPopover.hpp"
#include "TestingDeleteTestGroupPopover.hpp"
#include "TestingFailedView.hpp"
#include "TestingMoveToTestGroupPopover.hpp"
#include "TestingNewTestGroupPopover.hpp"
#include "TestingRenameTestGroupPopover.hpp"
#include "TestingRunTestsPopover.hpp"

namespace optifol
{

/**
 * @class TestingArea
 * @brief Manage the <i>Testing and Compliance</i> area
 *
 * @details
 *  The <i>Testing and Compliance</i> area provides controls for aggregating existing Requirement objects into
 *  TestGroup objects, and then executing unit test frameworks over the groups. The results of
 * Requirement-wise unit tests can be reviewed in the area, or exported to a report using the <i>Releases and
 * Reports</i> area capabilities. The following GTK elements are expected from the given Gtk::Builder: <table>
 *      <tr>
 *          <th>GTK C++ Class</th>
 *          <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Widget (abstract)</td>
 *          <td><code>testing_advice_unselected</code></td>
 *          <td>Advice to display when the area is unavailable</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Widget (abstract)</td>
 *          <td><code>testing_content</code></td>
 *          <td>Replacement to <code>testing_advice_unselected</code>, containing all active content</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnView</td>
 *          <td><code>test_groups_view</code></td>
 *          <td>Table to display nested TestGroup content</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>run_tests_popover</code></td>
 *          <td>Popover for executing the Test items in the selected TestGroup</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>run_tests</code></td>
 *          <td>Button for executing the external testing framework on the selected TestGroup</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>test_requirement_name</code></td>
 *          <td>Table column to display the name of the Requirement associated with the Test</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
```

```
 *          <td><code>test_target_executable</code></td>
 *          <td>Table column to display the target executable of the Test</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>test_fixture</code></td>
 *          <td>Table column to display the test fixture of the Test</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>test_status</code></td>
 *          <td>Table column to display the iconised result of the lastest Test run</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>new_test_group_popover</code></td>
 *          <td>Popover for creating a new TestGroup</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>new_test_group</code></td>
 *          <td>Button for opening <code>new_test_group_popover</code></td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>copy_to_test_group_popover</code></td>
 *          <td>Popover for copying a Requirement to a new TestGroup</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>copy_to_test_group</code></td>
 *          <td>Button for opening <code>copy_to_test_group_popover</code></td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>move_to_test_group_popover</code></td>
 *          <td>Popover for moving a Requirement between TestGroup objects</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>move_to_test_group</code></td>
 *          <td>Button for opening <code>move_to_test_group_popover</code></td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>delete_test_group_popover</code></td>
 *          <td>Popover for deleting a TestGroup</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::MenuButton</td>
 *          <td><code>delete_test_group</code></td>
 *          <td>Button for opening <code>delete_test_group_popover</code></td>
 *      </tr>
 *  </table>
 *  A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in the
 * expected type instantiations.
 */
class TestingArea : public IWindowArea
{
public:
    /**
     * @brief Construct a new area manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    explicit TestingArea(Gtk::Builder &builder);

    void select_model(const Glib::RefPtr<Subsystem> &new_subsystem) override;

    void deselect_model() override;

    Subsystem *get_active_subsystem() noexcept override;

    const Subsystem *observe_active_subsystem() const noexcept override;

    Glib::RefPtr<Gtk::TreeListRow> get_selected_row() noexcept;

    Glib::RefPtr<const Gtk::TreeListRow> get_selected_row() const noexcept;

    /**
     * @brief Copies the ref-counted pointer to the selected TestGroup. If a Requirement or Test is selected,
```

```
     * the owning TestGroup is returned.
     * @return The owning TestGroup of the selected object.
     * @throws std::runtime_error There is no selected object.
     */
    Glib::RefPtr<TestGroup> get_selected_test_group();

    /**
     * @brief Copies the ref-counted pointer to the selected Requirement. If a Test is selected, the owning
     * Requirement is returned.
     * @return The owning Requirement of the selected Test.
     * @throws std::runtime_error There is no selected object, or the selected object is unsuitable.
     */
    Glib::RefPtr<Requirement> get_selected_requirement();

private:
    /**
     * @brief Format a string encoding the TestResult passed state and execution time.
     * @param result An owning container of the TestResult to format.
     * @return A string describing the TestResult passed state and execution time.
     * @note The semantics of this function is poor. Ownership of the argument is not shared by the function.
     * It is required by the calling conventions of GTKmm. Ditto for the return type being encoded in @ref
     * std::optional.
     * @post The return value is such that @ref std::optional::has_value returns <code>true</code>.
     */
    static std::optional<Glib::ustring> bind_test_result(const std::shared_ptr<TestResult> &result) noexcept;

    /**
     * @brief Configure Gtk::ColumnViewColumn objects in the @ref test_groups_view.
     */
    void configure_columns() const;

    /**
     * @brief Configure the @ref selection_model.
     */
    void configure_selection_model() const;

    static const log4cxx::LoggerPtr area_logger;
    static const char *const area_name;

    Gtk::ColumnView *const test_groups_view;
    ContextButtonCorrespondence context_menu;
    std::pair<Gtk::Widget *, Gtk::Widget *> on_off_widgets;

    Glib::RefPtr<Subsystem> active_subsystem;
    Glib::RefPtr<Gtk::SingleSelection> selection_model = Gtk::SingleSelection::create();
    Glib::RefPtr<Gtk::TreeListModel> tree_model;

    TestingFailedView testing_failed_view;

    TestingNewTestGroupPopover new_test_group_popover;
    TestingRenameTestGroupPopover rename_test_group_popover;
    TestingDeleteTestGroupPopover delete_test_group_popover;
    TestingCopyToTestGroupPopover copy_requirement_popover;
    TestingMoveToTestGroupPopover move_requirement_popover;
    TestingRunTestsPopover run_tests_popover;
};

} // namespace optifol

#endif // TESTINGAREA_HPP
```

1.8.18.3 TestingCopyMovePopoverBase.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Testing and Compliance copying/moving popover base
 * @author Oliver Dixon
 * @date 2025-08-09
 * @version Development
 */

#include "TestingCopyMovePopoverBase.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
```

```cpp
#include "TestingArea.hpp"

namespace optifol
{

const char *const TestingCopyMovePopoverBase::popover_name = "Copy/Move Requirement to Test Group Popover";
const log4cxx::LoggerPtr TestingCopyMovePopoverBase::popover_logger =
        Logging::get_logger({"GUI", "TestingCompliance", "CopyMoveRequirement"});

TestingCopyMovePopoverBase::TestingCopyMovePopoverBase(TestingArea &testing_area, Gtk::Popover *my_popover,
        Gtk::Entry *requirement_entry, Gtk::DropDown *new_test_group_dropdown,
        Gtk::Button *const confirm_button, Gtk::Button *const cancel_button) :
    testing_area(testing_area),
    my_popover(my_popover),
    new_test_group_dropdown(new_test_group_dropdown),
    requirement_entry(requirement_entry)
{
    // Set up buttons and self.
    my_popover->signal_show().connect(sigc::mem_fun(*this, &TestingCopyMovePopoverBase::popover_show));
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingCopyMovePopoverBase::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingCopyMovePopoverBase::cancel_button_clicked));

    // Set up dropdown factory.
    new_test_group_dropdown->set_factory(
            configure_combo_box_factory(sigc::ptr_fun(&TestGroup::bind_name_to_label)));
}

void TestingCopyMovePopoverBase::popover_show() const noexcept
{
    // Setup the default destination test group dropdown.
    const auto test_group_model = testing_area.get_active_subsystem()->get_test_groups();
    if (new_test_group_dropdown->get_model() != test_group_model)
        new_test_group_dropdown->set_model(test_group_model);
    new_test_group_dropdown->set_selected(0);

    // Get the selected requirement. If a test is selected, use its parent requirement by walking the tree.
    try {
        const auto selected_requirement = testing_area.get_selected_requirement();
        requirement_entry->set_text(selected_requirement->property_name().get_value());
    } catch (const std::runtime_error &exception) {
        popover_logger->error(exception.what());
        my_popover->popdown();
    }
}

void TestingCopyMovePopoverBase::cancel_button_clicked() const noexcept
{
    my_popover->popdown();
}

Glib::RefPtr<Gtk::SignalListItemFactory> TestingCopyMovePopoverBase::configure_combo_box_factory(
        sigc::slot<void(const Glib::RefPtr<Gtk::ListItem> &)> &&bind_function)
{
    const auto factory = Gtk::SignalListItemFactory::create();

    factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
    factory->signal_bind().connect(std::move(bind_function));

    return factory;
}

} // namespace optifol
```

1.8.18.4 TestingCopyMovePopoverBase.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Testing and Compliance copying/moving popover base
 * @author Oliver Dixon
 * @date 2025-08-09
 * @version Development
 */
```

```cpp
#ifndef OPTIFOL_TESTINGCOPYMOVEPOPOVERBASE_HPP
#define OPTIFOL_TESTINGCOPYMOVEPOPOVERBASE_HPP

#include <gtkmm/button.h>
#include <gtkmm/dropdown.h>
#include <gtkmm/entry.h>
#include <gtkmm/popover.h>
#include <gtkmm/signallistitemfactory.h>
#include <log4cxx/logger.h>

namespace optifol
{

class TestingArea;
class Requirement;
class TestGroup;

/**
 * @class TestingCopyMovePopoverBase
 * @brief Provides a common base for GTK popovers providing functionality to copy or move Requirement objects
 * between TestGroup objects. Default callbacks are provided, but inheritors must implement the <i>Confirm</i>
 * action.
 */
class TestingCopyMovePopoverBase : public sigc::trackable
{
public:
    /**
     * @brief Destruct the TestingCopyMovePopoverBase instance.
     */
    // ReSharper disable once CppHidingFunction
    virtual ~TestingCopyMovePopoverBase() = default;

protected:
    /**
     * @brief Construct a new TestingCopyMovePopoverBase with the given GUI GTK elements.
     * @param testing_area The parental area.
     * @param my_popover The managed popover.
     * @param requirement_entry The read-only entry for the selected Requirement name.
     * @param new_test_group_dropdown The selection control for the target TestGroup.
     * @param confirm_button The button to confirm the action.
     * @param cancel_button The button to cancel the action.
     */
    TestingCopyMovePopoverBase(TestingArea &testing_area, Gtk::Popover *my_popover,
            Gtk::Entry *requirement_entry, Gtk::DropDown *new_test_group_dropdown,
            Gtk::Button *confirm_button, Gtk::Button *cancel_button);

    /**
     * @brief Display the popover by setting context-sensitive default selections for the drop-downs.
     * @see @ref set_test_group_dropdown for configuration of the <i>Current Test Group</i> drop-down.
     * @see @ref set_new_test_group_dropdown for configuration of the <i>New Test Group</i> drop-down.
     * @see @ref set_requirement_dropdown for configuration of the <i>Requirement Name</i> drop-down.
     */
    void popover_show() const noexcept;

    /**
     * @brief Handle a click of the <i>Confirm</i> button.
     */
    virtual void confirm_button_clicked() const noexcept = 0;

    /**
     * @brief Handle a click of the <i>Cancel</i> button by hiding the popover.
     */
    void cancel_button_clicked() const noexcept;

    /**
     * @brief Creates, configures, and returns a GTK factory for a Gtk::DropDown with a given bind function.
     * @param bind_function The handler for setting the contents of the Gtk::DropDown element for the given
     *  Gtk::ListItem.
     * @return The configured factory.
     */
    static Glib::RefPtr<Gtk::SignalListItemFactory> configure_combo_box_factory(
            sigc::slot<void(const Glib::RefPtr<Gtk::ListItem> &)> &&bind_function);

    TestingArea &testing_area;
    Gtk::Popover *const my_popover;
    Gtk::DropDown *const new_test_group_dropdown;

private:
    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;
```

```
    Gtk::Entry *const requirement_entry;
};

} // namespace optifol

#endif // OPTIFOL_TESTINGCOPYMOVEPOPOVERBASE_HPP
```

1.8.18.5 TestingCopyToTestGroupPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Testing and Compliance <i>Copy Requirement to Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-08
 * @version Development
 */

#include "TestingCopyToTestGroupPopover.hpp"
#include "../GTKHelpers.hpp"
#include "TestingArea.hpp"

namespace optifol
{

const char *const TestingCopyToTestGroupPopover::popover_name = "Copy Requirement to Test Group Popover";
const log4cxx::LoggerPtr TestingCopyToTestGroupPopover::popover_logger =
        Logging::get_logger({"GUI", "TestingCompliance", "CopyMoveRequirement", "Copy"});

TestingCopyToTestGroupPopover::TestingCopyToTestGroupPopover(
        Gtk::Builder &builder, TestingArea &testing_area) :
    TestingCopyMovePopoverBase(testing_area,
            GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "copy_to_test_group_popover"),
            GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "copy_to_test_group_requirement"),
            GTKHelpers::get_widget<Gtk::DropDown>(popover_name, builder, "copy_to_test_group_new_group"),
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "copy_to_test_group_confirm"),
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "copy_to_test_group_cancel"))
{
}

void TestingCopyToTestGroupPopover::confirm_button_clicked() const noexcept
{
    my_popover->popdown();

    try {
        const auto requirement = testing_area.get_selected_requirement();
        const auto target_test_group =
                dynamic_cast<TestGroup *>(new_test_group_dropdown->get_selected_item().get());

        if (requirement == nullptr) {
            popover_logger->warn("Not copying Requirement, as no Requirement selected.");
            return;
        }

        if (target_test_group == nullptr) {
            popover_logger->warn("Not copying Requirement, as no destination Test Group selected.");
            return;
        }

        target_test_group->insert_object(requirement);
        popover_logger->debug("Copied Requirement \"" + requirement->property_name().get_value() +
                "\" into Test Group \"" + target_test_group->property_name().get_value() + "\".");
    } catch (const std::runtime_error &exception) {
        popover_logger->error("Not copying Requirement: " + std::string(exception.what()));
    }
}


} // namespace optifol
```

1.8.18.6 TestingCopyToTestGroupPopover.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Testing and Compliance <i>Copy Requirement to Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-08
 * @version Development
 */

#ifndef OPTIFOL_TESTINGCOPYTOTESTGROUPPOPOVER_HPP
#define OPTIFOL_TESTINGCOPYTOTESTGROUPPOPOVER_HPP

#include <gtkmm/builder.h>

#include "TestingCopyMovePopoverBase.hpp"

namespace optifol
{

class TestingArea;

/**
 * @class TestingCopyToTestGroupPopover
 * @brief Manage the <i>Copy Requirement to Test Group</i> popover for the TestingArea.
 * @see TestingArea for the parent area.
 *
 * @details
 *  The <i>Copy Requirement to Test Group</i> popover provides controls for copying an existing Requirement in
 * a TestGroup to another existing TestGroup. The control automatically populates its entries with likely
 * defaults using the TestGroup/Requirement/Test models from the parental TestArea. The following GTK elements
 * are expected from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th> <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>copy_to_test_group_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>copy_to_test_group_confirm</code></td>
 *          <td>Confirm copy of Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>copy_to_test_group_cancel</code></td>
 *          <td>Cancels copy of Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>copy_to_test_group_requirement</code></td>
 *          <td>Reported name of Requirement to copy</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::DropDown</td>
 *          <td><code>copy_to_test_group_new_group</code></td>
 *          <td>Selection of destination TestGroup/move</td>
 *      </tr>
 *  </table>
 *  A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in the
 * expected type instantiations.
 */
class TestingCopyToTestGroupPopover : public TestingCopyMovePopoverBase
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param testing_area A mutating reference to the TestingArea of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    TestingCopyToTestGroupPopover(Gtk::Builder &builder, TestingArea &testing_area);

private:
    /**
     * @copybrief TestingCopyMovePopoverBase::confirm_button_clicked
     * @details Copies the Requirement from its current TestGroup to the selected destination TestGroup. The
```

```
     * source TestGroup retains its copy of the Requirement.
     */
    void confirm_button_clicked() const noexcept override;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;
};

} // namespace optifol

#endif // OPTIFOL_TESTINGCOPYTOTESTGROUPPOPOVER_HPP
```

1.8.18.7 TestingDeleteTestGroupPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Testing and Compliance <i>Delete Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-09
 * @version Development
 */

#include "TestingDeleteTestGroupPopover.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "TestingArea.hpp"

namespace optifol
{

const char *const TestingDeleteTestGroupPopover::popover_name = "Delete Test Group Popover";
const log4cxx::LoggerPtr TestingDeleteTestGroupPopover::popover_logger =
        Logging::get_logger({"GUI", "TestingCompliance", "DeleteTestGroup"});

TestingDeleteTestGroupPopover::TestingDeleteTestGroupPopover(
        Gtk::Builder &builder, TestingArea &testing_area) :
    testing_area(testing_area),
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "delete_test_group_popover")),
    test_group_entry(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "delete_test_group_name"))
{
    // Get extra elements needed only for the constructor lifetime.
    const auto confirm_button =
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "delete_test_group_confirm");
    const auto cancel_button =
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "delete_test_group_cancel");

    // Set up buttons and self.
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingDeleteTestGroupPopover::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingDeleteTestGroupPopover::cancel_button_clicked));
    my_popover->signal_show().connect(sigc::mem_fun(*this, &TestingDeleteTestGroupPopover::popover_shown));
}

void TestingDeleteTestGroupPopover::confirm_button_clicked() const noexcept
{
    try {
        const auto target_group = testing_area.get_selected_test_group();
        const auto groups = testing_area.get_active_subsystem()->get_test_groups().get();
        const auto group_count = groups->get_n_items();

        /*
         * Linear search is acceptable; we maintain a relatively small number of test groups, and it's not
         * worth the additional overhead of maintaining an index hash table.
         */
        for (guint group_index = 0; group_index < group_count; ++group_index)
            if (groups->get_item(group_index) == target_group) {
                groups->remove(group_index);
                popover_logger->debug(
                        "Removed Test Group \"" + target_group->property_name().get_value() + "\".");
                break;
            }
    } catch (const std::runtime_error &selection_error) {
        popover_logger->error("Could not discover the selected Test Group entry.");
```

```
        popover_logger->error(selection_error.what());
    }

    my_popover->popdown();
}

void TestingDeleteTestGroupPopover::cancel_button_clicked() const noexcept
{
    my_popover->popdown();
}

void TestingDeleteTestGroupPopover::popover_shown() const noexcept
{
    try {
        test_group_entry->set_text(testing_area.get_selected_test_group()->property_name().get_value());
    } catch (const std::runtime_error &selection_error) {
        my_popover->popdown();
        popover_logger->error("Could not discover the selected Test Group entry.");
        popover_logger->error(selection_error.what());
    }
}

} // namespace optifol
```

1.8.18.8 TestingDeleteTestGroupPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Testing and Compliance <i>Delete Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-09
 * @version Development
 */

#ifndef OPTIFOL_TESTINGDELETETESTGROUPPOPOVER_HPP
#define OPTIFOL_TESTINGDELETETESTGROUPPOPOVER_HPP

#include <gtkmm/builder.h>
#include <gtkmm/entry.h>
#include <gtkmm/popover.h>
#include <log4cxx/logger.h>

namespace optifol
{

class TestingArea;

/**
 * @class TestingDeleteTestGroupPopover
 * @brief Manage the <i>Delete Test Group</i> popover for the Testing and Compliance area.
 * @see TestingArea for the parent area
 *
 * @details
 *  The <i>Delete Test Group</i> popover provides controls for deleting an existing TestGroup. The following
 * GTK elements are expected to be available from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th>
 *          <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>delete_test_group_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>delete_test_group_confirm</code></td>
 *          <td>Confirm deletion button</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>delete_test_group_cancel</code></td>
 *          <td>Cancel deletion button</td>
 *      </tr>
 *      <tr>
```

```
 *          <td>Gtk::Entry</td>
 *          <td><code>delete_test_group_name</code></td>
 *          <td>Read-only entry to hold the name of the TestGroup to delete</td>
 *      </tr>
 *  </table>
 */
class TestingDeleteTestGroupPopover : public sigc::trackable
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param testing_area A mutating reference to the TestingArea of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    TestingDeleteTestGroupPopover(Gtk::Builder &builder, TestingArea &testing_area);

private:
    /**
     * @brief Handle a click of the <i>Confirm</i> button by deleting the selected TestGroup.
     */
    void confirm_button_clicked() const noexcept;

    /**
     * @brief Handle a click of the <i>Cancel</i> button by hiding the popover.
     */
    void cancel_button_clicked() const noexcept;

    /**
     * @brief Handle a show of the popover by displaying the selected TestGroup name.
     */
    void popover_shown() const noexcept;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    TestingArea &testing_area;

    Gtk::Popover *const my_popover;
    Gtk::Entry *const test_group_entry;
};

} // namespace optifol

#endif // OPTIFOL_TESTINGDELETETESTGROUPPOPOVER_HPP
```

1.8.18.9 TestingFailedView.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Testing and Compliance Failed Test Results view
 * @author Oliver Dixon
 * @date 2025-08-10
 * @version Development
 */

#include "TestingFailedView.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "../Storage/Subsystem.hpp"

namespace optifol
{

const log4cxx::LoggerPtr TestingFailedView::area_logger =
        Logging::get_logger({"GUI", "TestingCompliance", "FailedTests"});
const char *const TestingFailedView::area_name = "Testing and Compliance Failed Tests Area";

TestingFailedView::TestingFailedView(Gtk::Builder &builder) :
    table(GTKHelpers::get_widget<Gtk::ColumnView>(area_name, builder, "test_groups_failed_view")),
    container(GTKHelpers::get_widget<Gtk::Widget>(area_name, builder, "test_groups_failed_container"))
{
    table->set_model(selection_model);
    configure_columns();
```

```
}

Subsystem *TestingFailedView::get_active_subsystem() noexcept
{
    return active_subsystem.get();
}

const Subsystem *TestingFailedView::observe_active_subsystem() const noexcept
{
    return active_subsystem.get();
}

void TestingFailedView::select_model(const Glib::RefPtr<Subsystem> &new_subsystem)
{
    active_subsystem = new_subsystem;
    tree_model = Gtk::TreeListModel::create(
            active_subsystem->get_test_groups(), &ITestModelNode::get_given_results_tree, true, true);
    selection_model->set_model(tree_model);
    tree_model->signal_items_changed().connect(sigc::mem_fun(*this, &TestingFailedView::handle_model_change));
}

void TestingFailedView::deselect_model()
{
    active_subsystem = nullptr;
    tree_model = nullptr;
    selection_model->set_model(nullptr);
}

void TestingFailedView::configure_columns() const
{
    const auto columns = table->get_columns();
    const auto column_count = columns->get_n_items();

    for (guint position = 0; position < column_count; ++position) {
        Glib::RefPtr<Gtk::ColumnViewColumn> column = nullptr;

        if ((column = columns->get_typed_object<Gtk::ColumnViewColumn>(position)) != nullptr) {
            const auto &gtk_id = column->get_id();
            const auto factory = Gtk::SignalListItemFactory::create();

            if (gtk_id == "failed_test_message") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_expandable_label, false));
                factory->signal_bind().connect([this](const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept
                        { StorageObjectBase::bind_name_property_expandable(list_item, tree_model); });

            } else if (gtk_id == "failed_test_source_file") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(
                        [](const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept -> void
                        {
                            const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
                            const auto typed_result =
                                    dynamic_cast<const PartialTestResult *>(list_item->get_item().get());

                            if (label == nullptr || typed_result == nullptr)
                                return;

                            Glib::Binding::bind_property(typed_result->property_file(),
                                    label->property_label(), Glib::Binding::Flags::SYNC_CREATE);
                        });

            } else if (gtk_id == "failed_test_source_line") {

                factory->signal_setup().connect(sigc::bind(&GTKHelpers::setup_label, false));
                factory->signal_bind().connect(
                        [](const Glib::RefPtr<Gtk::ListItem> &list_item) noexcept -> void
                        {
                            const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
                            const auto typed_result =
                                    dynamic_cast<const PartialTestResult *>(list_item->get_item().get());

                            if (label == nullptr || typed_result == nullptr)
                                return;

                            Glib::Binding::bind_property(typed_result->property_line(),
                                    label->property_label(), Glib::Binding::Flags::SYNC_CREATE,
                                    [](const guint line) -> std::optional<Glib::ustring>
                                    { return std::to_string(line); });
                        });
```

```
            } else
                // Jump out here if unrecognised, so all further code can assume a factory was configured.
                continue;

            column->set_factory(factory);
        }
    }
}

void TestingFailedView::handle_model_change(
        const guint initial_index, const guint removed_count, const guint added_count) const noexcept
{
    std::ignore = initial_index;
    std::ignore = removed_count;
    std::ignore = added_count;

    /*
     * If the tree model doesn't exist, we always want to disable the failed view: no model means no failed
     * tests.
     *
     * Recall that the number of items in the failed model is lower-bounded by the number of test groups. Once
     * the number of items in the failed model reaches the number of test groups, we know that all test groups
     * must be empty. Since they are empty, there are no failure records, thus the container should be hidden.
     */
    const bool should_display = tree_model != nullptr &&
            tree_model->get_n_items() > active_subsystem->get_test_groups()->get_n_items();

    if (container->get_visible() != should_display)
        container->set_visible(should_display);
}

} // namespace optifol
```

1.8.18.10 TestingFailedView.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Testing and Compliance Failed Test Results view
 * @author Oliver Dixon
 * @date 2025-08-10
 * @version Development
 */

#ifndef OPTIFOL_TESTINGFAILEDVIEW_HPP
#define OPTIFOL_TESTINGFAILEDVIEW_HPP

#include <glibmm/refptr.h>
#include <gtkmm/builder.h>
#include <gtkmm/columnview.h>
#include <gtkmm/noselection.h>
#include <gtkmm/treelistmodel.h>
#include <log4cxx/logger.h>

#include "../IWindowArea.hpp"

namespace optifol
{

/**
 * @class TestingFailedView
 * @brief Manage the <i>Failed Tests View</i> section of the <i>Testing and Compliance</i> area.
 *
 * @details
 *  The <i>Failed Tests View</i> presents a read-only tabular representation of PartialTestResult objects,
 * grouped by their respective owning Test objects, which are in turn grouped by the TestGroup objects. The
 * following GTK elements are expected from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th>
 *          <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Widget (abstract)</td>
 *          <td><code>test_groups_failed_container</code></td>
 *          <td>Containing widget of the entire sub-area.</td>
```

```
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnView</td>
 *          <td><code>test_groups_failed_view</code></td>
 *          <td>Table detained by <code>test_groups_failed_container</code> tabulating PartialTestResult
 * fields.</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>failed_test_message</code></td>
 *          <td>Table column to display the message of the PartialTestResult, or the name of the Test
 * thereof.</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>failed_test_source_file</code></td>
 *          <td>Table column to display the source file name of the PartialTestResult.</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::ColumnViewColumn</td>
 *          <td><code>failed_test_source_line</code></td>
 *          <td>Table column to display the source line number of the PartialTestResult.</td>
 *      </tr>
 *  </table>
 *  A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in the
 * expected type instantiations.
 */
class TestingFailedView : public IWindowArea
{
public:
    /**
     * @brief Construct a new sub-area manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    explicit TestingFailedView(Gtk::Builder &builder);

    Subsystem *get_active_subsystem() noexcept override;

    const Subsystem *observe_active_subsystem() const noexcept override;

    void select_model(const Glib::RefPtr<Subsystem> &new_subsystem) override;

    void deselect_model() override;

private:
    void configure_columns() const;

    /**
     * @brief Handles a change in the model. In particular, show or hide the entire @ref container viewing
     * pane.
     * @param initial_index The initial index of the model changes.
     * @param removed_count The number of removed items.
     * @param added_count The number of added items.
     */
    void handle_model_change(guint initial_index, guint removed_count, guint added_count) const noexcept;

    static const log4cxx::LoggerPtr area_logger;
    static const char *const area_name;

    Gtk::Widget *const container;
    Gtk::ColumnView *const table;

    Glib::RefPtr<Subsystem> active_subsystem;
    Glib::RefPtr<Gtk::NoSelection> selection_model = Gtk::NoSelection::create();
    Glib::RefPtr<Gtk::TreeListModel> tree_model;
};

} // namespace optifol

#endif // OPTIFOL_TESTINGFAILEDVIEW_HPP
```

1.8.18.11 TestingMoveToTestGroupPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */
```

```
/**
 * @file
 * @brief Class implementation for the Testing and Compliance <i>Move Requirement to Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-09
 * @version Development
 */

#include "TestingMoveToTestGroupPopover.hpp"
#include "../GTKHelpers.hpp"
#include "TestingArea.hpp"

namespace optifol
{

const char *const TestingMoveToTestGroupPopover::popover_name = "Move Requirement to Test Group Popover";
const log4cxx::LoggerPtr TestingMoveToTestGroupPopover::popover_logger =
        Logging::get_logger({"GUI", "TestingCompliance", "CopyMoveRequirement", "Move"});

TestingMoveToTestGroupPopover::TestingMoveToTestGroupPopover(
        Gtk::Builder &builder, TestingArea &testing_area) :
    TestingCopyMovePopoverBase(testing_area,
            GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "move_to_test_group_popover"),
            GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "move_to_test_group_requirement"),
            GTKHelpers::get_widget<Gtk::DropDown>(popover_name, builder, "move_to_test_group_new_group"),
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "move_to_test_group_confirm"),
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "move_to_test_group_cancel"))
{
}

void TestingMoveToTestGroupPopover::confirm_button_clicked() const noexcept
{
    my_popover->popdown();

    try {
        const auto source_test_group = testing_area.get_selected_test_group();
        const auto requirement = testing_area.get_selected_requirement();
        const auto target_test_group =
                dynamic_cast<TestGroup *>(new_test_group_dropdown->get_selected_item().get());

        if (source_test_group == nullptr) {
            popover_logger->warn("Not moving Requirement, as no source Test Group selected.");
            return;
        }

        if (requirement == nullptr) {
            popover_logger->warn("Not moving Requirement, as no Requirement selected.");
            return;
        }

        if (target_test_group == nullptr) {
            popover_logger->warn("Not moving Requirement, as no destination Test Group selected.");
            return;
        }

        target_test_group->insert_object(requirement); // Do this first, in case it throws an exception.
        source_test_group->delete_object(requirement);

        popover_logger->debug("Moved Requirement \"" + requirement->property_name().get_value() +
                "\" from Test Group \"" + source_test_group->property_name().get_value() +
                "\" into Test Group \"" + target_test_group->property_name().get_value() + "\".");
    } catch (const std::runtime_error &exception) {
        popover_logger->error("Not moving Requirement: " + std::string(exception.what()));
    }
}

} // namespace optifol
```

1.8.18.12 TestingMoveToTestGroupPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Testing and Compliance <i>Move Requirement to Test Group</i> popover
 * @author Oliver Dixon
```

```
 * @date 2025-08-09
 * @version Development
 */

#ifndef OPTIFOL_TESTINGMOVETOTESTGROUPPOPOVER_HPP
#define OPTIFOL_TESTINGMOVETOTESTGROUPPOPOVER_HPP

#include <gtkmm/builder.h>

#include "TestingCopyMovePopoverBase.hpp"

namespace optifol
{

/**
 * @class TestingMoveToTestGroupPopover
 * @brief Manage the <i>Move Requirement to Test Group</i> popover for the TestingArea.
 * @see TestingArea for the parent area.
 *
 * @details
 *  The <i>Move Requirement to Test Group</i> popover provides controls for moving an existing Requirement
 * between TestGroup objects. The control automatically populates its entries with likely defaults using the
 * TestGroup/ Requirement/Test models from the parental TestArea. The following GTK elements are expected from
 * the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th> <th>Unique Identifier</th> <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>move_to_test_group_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>move_to_test_group_confirm</code></td>
 *          <td>Confirm move of Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>move_to_test_group_cancel</code></td>
 *          <td>Cancels move of Requirement</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>move_to_test_group_requirement</code></td>
 *          <td>Reported name of Requirement to move</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::DropDown</td>
 *          <td><code>move_to_test_group_new_group</code></td>
 *          <td>Selection of destination TestGroup/move</td>
 *      </tr>
 *  </table>
 *  A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in the
 * expected type instantiations.
 */
class TestingMoveToTestGroupPopover : public TestingCopyMovePopoverBase
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param testing_area A mutating reference to the TestingArea of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    TestingMoveToTestGroupPopover(Gtk::Builder &builder, TestingArea &testing_area);

private:
    /**
     * @copybrief TestingCopyMovePopoverBase::confirm_button_clicked
     * @details Moves the Requirement from its current TestGroup to the selected destination TestGroup. The
     * source TestGroup does not retain its copy of the Requirement.
     */
    void confirm_button_clicked() const noexcept override;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;
};

} // namespace optifol

#endif // OPTIFOL_TESTINGMOVETOTESTGROUPPOPOVER_HPP
```

1.8.18.13 TestingNewTestGroupPopover.cpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Testing and Compliance <i>New Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-08
 * @version Development
 */

#include "TestingNewTestGroupPopover.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "TestingArea.hpp"

namespace optifol
{

const char *const TestingNewTestGroupPopover::popover_name = "New Test Group Popover";
const log4cxx::LoggerPtr TestingNewTestGroupPopover::popover_logger =
        Logging::get_logger({"GUI", "TestingCompliance", "NewTestGroup"});

TestingNewTestGroupPopover::TestingNewTestGroupPopover(Gtk::Builder &builder, TestingArea &testing_area) :
    testing_area(testing_area),
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "new_test_group_popover")),
    confirm_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "new_test_group_confirm")),
    name_entry(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "new_test_group_name"))
{
    // Get extra elements needed only for the constructor lifetime.
    const auto cancel_button =
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "new_test_group_cancel");

    // Set up buttons and self.
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingNewTestGroupPopover::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingNewTestGroupPopover::cancel_button_clicked));
    my_popover->signal_show().connect(sigc::mem_fun(*this, &TestingNewTestGroupPopover::popover_shown));
    name_entry->signal_changed().connect(
            sigc::mem_fun(*this, &TestingNewTestGroupPopover::name_entry_changed));
}

void TestingNewTestGroupPopover::confirm_button_clicked() const
{
    my_popover->popdown();
    testing_area.observe_active_subsystem()->get_test_groups()->append(
            Glib::make_refptr_for_instance(new TestGroup(name_entry->get_text())));

    popover_logger->debug("Created new Test Group with name \"" + name_entry->get_text() + "\".");
}

void TestingNewTestGroupPopover::cancel_button_clicked() const noexcept
{
    my_popover->popdown();
}

void TestingNewTestGroupPopover::popover_shown() const noexcept
{
    clear_inputs();
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from popover show callback.
void TestingNewTestGroupPopover::clear_inputs() const noexcept
{
    confirm_button->set_sensitive(false);
    name_entry->set_text("");
}

void TestingNewTestGroupPopover::name_entry_changed() const noexcept
{
    confirm_button->set_sensitive(!name_entry->get_text().empty());
}

} // namespace optifol
```

1.8.18.14 TestingNewTestGroupPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Testing and Compliance <i>New Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-08
 * @version Development
 */

#ifndef OPTIFOL_TESTINGNEWTESTGROUPPOPOVER_HPP
#define OPTIFOL_TESTINGNEWTESTGROUPPOPOVER_HPP

#include <gtkmm/builder.h>
#include <gtkmm/button.h>
#include <gtkmm/entry.h>
#include <gtkmm/popover.h>
#include <log4cxx/logger.h>

namespace optifol
{

class TestingArea;

/**
 * @class TestingNewTestGroupPopover
 * @brief Manage the <i>New Test Group</i> popover for the Testing and Compliance area.
 * @see TestingArea for the parent area
 *
 * @details
 *  The <i>New Test Group</i> popover provides controls for creating a new TestGroup to hold Requirement
 * objects, which in turn detain one or more Test entities. The following GTK elements are expected to be
 * available from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th> <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>new_test_group_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>new_test_group_confirm</code></td>
 *          <td>Confirm changes button</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>new_test_group_cancel</code></td>
 *          <td>Cancel changes button</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>new_test_group_name</code></td>
 *          <td>Entry for the name of the new TestGroup</td>
 *      </tr>
 *  </table>
 */
class TestingNewTestGroupPopover : public sigc::trackable
{
public:
    TestingNewTestGroupPopover(Gtk::Builder &builder, TestingArea &testing_area);

private:
    void confirm_button_clicked() const;

    void cancel_button_clicked() const noexcept;

    void popover_shown() const noexcept;

    void clear_inputs() const noexcept;

    void name_entry_changed() const noexcept;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;
```

```
    TestingArea &testing_area;

    Gtk::Popover *const my_popover;
    Gtk::Button *const confirm_button;
    Gtk::Entry *const name_entry;
};

} // namespace optifol

#endif // OPTIFOL_TESTINGNEWTESTGROUPPOPOVER_HPP
```

1.8.18.15 TestingRenameTestGroupPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Testing and Compliance <i>Rename Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-10
 * @version Development
 */

#include "TestingRenameTestGroupPopover.hpp"
#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "TestingArea.hpp"

namespace optifol
{

const char *const TestingRenameTestGroupPopover::popover_name = "Rename Test Group Popover";
const log4cxx::LoggerPtr TestingRenameTestGroupPopover::popover_logger =
        Logging::get_logger({"GUI", "TestingCompliance", "RenameTestGroup"});

TestingRenameTestGroupPopover::TestingRenameTestGroupPopover(
        Gtk::Builder &builder, TestingArea &testing_area) :
    testing_area(testing_area),
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "rename_test_group_popover")),
    confirm_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "rename_test_group_confirm")),
    old_name_entry(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "rename_test_group_old_name")),
    new_name_entry(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "rename_test_group_new_name"))
{
    // Get extra elements needed only for the constructor lifetime.
    const auto cancel_button =
            GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "rename_test_group_cancel");

    // Set up buttons and self.
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingRenameTestGroupPopover::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingRenameTestGroupPopover::cancel_button_clicked));
    my_popover->signal_show().connect(sigc::mem_fun(*this, &TestingRenameTestGroupPopover::popover_shown));
    new_name_entry->signal_changed().connect(
            sigc::mem_fun(*this, &TestingRenameTestGroupPopover::new_name_changed));
}

void TestingRenameTestGroupPopover::confirm_button_clicked() const noexcept
{
    try {
        const auto existing_group = testing_area.get_selected_test_group();
        existing_group->property_name().set_value(new_name_entry->get_text());
        popover_logger->debug("Renamed Test Group \"" + old_name_entry->get_text() + "\" to \"" +
                existing_group->property_name().get_value() + "\".");
    } catch (const std::runtime_error &selection_error) {
        popover_logger->error("Could not discover the selected Test Group entry.");
        popover_logger->error(selection_error.what());
    }

    my_popover->popdown();
    clear_inputs();
}

void TestingRenameTestGroupPopover::cancel_button_clicked() const noexcept
{
```

```
    my_popover->popdown();
    clear_inputs();
}

void TestingRenameTestGroupPopover::popover_shown() const noexcept
{
    try {
        old_name_entry->set_text(testing_area.get_selected_test_group()->property_name().get_value());
    } catch (const std::runtime_error &selection_error) {
        my_popover->popdown();
        popover_logger->error("Could not discover the selected Test Group entry.");
        popover_logger->error(selection_error.what());
    }
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive. Invoked from popover callbacks.
void TestingRenameTestGroupPopover::clear_inputs() const noexcept
{
    new_name_entry->set_text("");
}

void TestingRenameTestGroupPopover::new_name_changed() const noexcept
{
    const bool should_enable = new_name_entry->get_text_length() > 0;
    if (new_name_entry->get_sensitive() != should_enable)
        new_name_entry->set_sensitive(should_enable);
}

} // namespace optifol
```

1.8.18.16 TestingRenameTestGroupPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Testing and Compliance <i>Rename Test Group</i> popover
 * @author Oliver Dixon
 * @date 2025-08-10
 * @version Development
 */

#ifndef OPTIFOL_TESTINGRENAMETESTGROUPPOPOVER_HPP
#define OPTIFOL_TESTINGRENAMETESTGROUPPOPOVER_HPP

#include <gtkmm/builder.h>
#include <gtkmm/button.h>
#include <gtkmm/entry.h>
#include <gtkmm/popover.h>
#include <log4cxx/logger.h>

namespace optifol
{

class TestingArea;

/**
 * @class TestingRenameTestGroupPopover
 * @brief Manage the <i>Rename Test Group</i> popover for the Testing and Compliance area.
 * @see TestingArea for the parent area
 *
 * @details
 *  The <i>Rename Test Group</i> popover provides controls for renaming an existing TestGroup. The following
 * GTK elements are expected to be available from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th>
 *          <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>rename_test_group_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>rename_test_group_confirm</code></td>
 *          <td>Confirm deletion button</td>
```

```
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>rename_test_group_cancel</code></td>
 *          <td>Cancel deletion button</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>rename_test_group_old_name</code></td>
 *          <td>Read-only entry to hold the name of the existing TestGroup to rename</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>rename_test_group_old_name</code></td>
 *          <td>Entry to accept the new name of the TestGroup</td>
 *      </tr>
 *  </table>
 */
class TestingRenameTestGroupPopover : public sigc::trackable
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param testing_area A mutating reference to the TestingArea of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    TestingRenameTestGroupPopover(Gtk::Builder &builder, TestingArea &testing_area);

private:
    /**
     * @brief Handle a click of the <i>Confirm</i> button by renaming the selected TestGroup.
     */
    void confirm_button_clicked() const noexcept;

    /**
     * @brief Handle a click of the <i>Cancel</i> button by hiding the popover.
     */
    void cancel_button_clicked() const noexcept;

    /**
     * @brief Handle a show of the popover by displaying the selected TestGroup name.
     */
    void popover_shown() const noexcept;

    /**
     * @brief Clear all user-provided inputs.
     */
    void clear_inputs() const noexcept;

    /**
     * @brief React to a change in the @ref new_name_entry contents; in particular, enable or disable the
     * <i>Confirm</i> button.
     */
    void new_name_changed() const noexcept;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    TestingArea &testing_area;

    Gtk::Popover *const my_popover;
    Gtk::Button *const confirm_button;
    Gtk::Entry *const old_name_entry;
    Gtk::Entry *const new_name_entry;
};

} // namespace optifol

#endif // OPTIFOL_TESTINGRENAMETESTGROUPPOPOVER_HPP
```

1.8.18.17 TestingRunTestsPopover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
```

```
 * @file
 * @brief Class implementation for the <i>Run Tests</i> popover in the <i>Testing and Compliance</i> area.
 * @author Oliver Dixon
 * @date 2025-07-27
 * @version Development
 */

#include "TestingRunTestsPopover.hpp"

#include "../GTKHelpers.hpp"
#include "../Logging.hpp"
#include "TestingArea.hpp"

namespace optifol
{

const char *const TestingRunTestsPopover::popover_name = "Run Tests Popover";
const log4cxx::LoggerPtr TestingRunTestsPopover::popover_logger =
        Logging::get_logger({"GUI", "TestingCompliance", "RunTests"});

TestingRunTestsPopover::TestingRunTestsPopover(Gtk::Builder &builder, TestingArea &testing_area) :
    testing_area(testing_area),
    my_popover(GTKHelpers::get_widget<Gtk::Popover>(popover_name, builder, "run_tests_popover")),
    confirm_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "run_tests_confirm")),
    cancel_button(GTKHelpers::get_widget<Gtk::Button>(popover_name, builder, "run_tests_cancel")),
    test_group_name_entry(GTKHelpers::get_widget<Gtk::Entry>(popover_name, builder, "test_group_name"))
{
    my_popover->signal_show().connect(sigc::mem_fun(*this, &TestingRunTestsPopover::show_popover));
    confirm_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingRunTestsPopover::confirm_button_clicked));
    cancel_button->signal_clicked().connect(
            sigc::mem_fun(*this, &TestingRunTestsPopover::cancel_button_clicked));
}

void TestingRunTestsPopover::confirm_button_clicked() noexcept
{
    const auto selected_test_group = testing_area.get_selected_test_group();
    const auto begin = selected_test_group->begin_execution_groups();
    const auto end = selected_test_group->end_execution_groups();

    for (std::remove_const_t<decltype(begin)> exe_group_it = begin; exe_group_it != end; ++exe_group_it)
        exe_group_it->get()->run();
}

void TestingRunTestsPopover::cancel_button_clicked() const noexcept
{
    my_popover->popdown();
    clear_inputs();
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from button-click callback.
void TestingRunTestsPopover::clear_inputs() const noexcept
{
    test_group_name_entry->set_text("");
}

void TestingRunTestsPopover::show_popover() const noexcept
{
    try {
        test_group_name_entry->set_text(testing_area.get_selected_test_group()->property_name().get_value());
    } catch (const std::runtime_error &selection_error) {
        popover_logger->error("Could not discover the selected Test Group entry.");
        popover_logger->error(selection_error.what());
    }
}

} // namespace optifol
```

1.8.18.18 TestingRunTestsPopover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the <i>Run Tests</i> popover in the <i>Testing and Compliance</i> area.
 * @author Oliver Dixon
```

```
 * @date 2025-07-27
 * @version Development
 */

#ifndef TESTINGRUNTESTSPOPOVER_HPP
#define TESTINGRUNTESTSPOPOVER_HPP

#include <gtkmm/builder.h>
#include <gtkmm/button.h>
#include <gtkmm/entry.h>
#include <gtkmm/popover.h>
#include <gtkmm/textview.h>
#include <log4cxx/logger.h>

namespace optifol
{

class TestGroup;
class TestListenerBase;
class TestingArea;

/**
 * @class TestingRunTestsPopover
 * @brief Manage the <i>Run Tests</i> popover for the <i>Testing and Compliance</i> area.
 * @see TestingArea for the parent area
 *
 * @details
 *  The <i>Run Tests</i> popover provides controls for executing unit test specifications through an external
 *  executable(s) for the selected TestGroup within the Subsystem. The following GTK elements are expected to
 * be available from the given Gtk::Builder: <table> <tr> <th>GTK C++ Class</th> <th>Unique Identifier</th>
 *          <th>Purpose</th>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Popover</td>
 *          <td><code>run_tests_popover</code></td>
 *          <td>Managed popover</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>run_tests_confirm</code></td>
 *          <td>Confirm execution of test executable</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Button</td>
 *          <td><code>run_tests_cancel</code></td>
 *          <td>Cancels execution of test executables</td>
 *      </tr>
 *      <tr>
 *          <td>Gtk::Entry</td>
 *          <td><code>test_group_name</code></td>
 *          <td>Read-only text area for the name of the selected TestGroup</td>
 *      </tr>
 *  </table>
 *  A @ref std::runtime_error will be thrown by the class constructor if any of these are inaccessible in the
 * expected type instantiations.
 */
class TestingRunTestsPopover : public sigc::trackable
{
public:
    /**
     * @brief Construct a new popover manager, registering callbacks on elements loaded by the given builder
     * @param builder A GTK builder containing popover UI elements
     * @param testing_area A mutating reference to the TestingArea of which the popover is a member
     * @throws std::runtime_error A required GTK element/widget could not be loaded from the given builder
     */
    TestingRunTestsPopover(Gtk::Builder &builder, TestingArea &testing_area);

private:
    /**
     * @brief Handle a click of the <i>Confirm</i> by attempting to create a Requirement with the given
     * characteristics.
     */
    void confirm_button_clicked() noexcept;

    /**
     * @brief Handle a click of the <i>Cancel</i> button by discarding all input and closing the popover.
     */
    void cancel_button_clicked() const noexcept;

    /**
     * @brief Clear all user fields in the popover
```

```
     */
    void clear_inputs() const noexcept;

    /**
     * @brief Handle the popover becoming visible.
     */
    void show_popover() const noexcept;

    static const char *const popover_name;
    static const log4cxx::LoggerPtr popover_logger;

    TestingArea &testing_area;

    Gtk::Popover *const my_popover;
    Gtk::Button *const confirm_button;
    Gtk::Button *const cancel_button;
    Gtk::Entry *const test_group_name_entry;
};

} // namespace optifol

#endif // TESTINGRUNTESTSPOPOVER_HPP
```

1.9 Inference

1.9.1 ExpressionFactory.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the CNF expression factory
 * @author Oliver Dixon
 * @date 2025-09-14
 * @version Development
 */

#include "ExpressionFactory.hpp"

#include "../Exceptions/SemanticException.hpp"
#include "../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"
#include "../IR/Sentences/SentenceRoot.hpp"
#include "../Logging.hpp"

#include "../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/DMLVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/DisjunctionDistributionVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/ImplicationEliminationVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/QuantifierExtractingVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/SkolemIntroducingVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/SymbolStandardisingVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/UniversalEliminationVisitor.hpp"

namespace optifol
{

const log4cxx::LoggerPtr ExpressionFactory::cnf_logger =
        Logging::get_logger({"LogicServices", "CNFNormalisation"});
TextSerialiserVisitor ExpressionFactory::serialiser_visitor;

std::unique_ptr<SentenceRoot> ExpressionFactory::build_sentence(
        std::unique_ptr<MutableSentenceRoot> &&sentence_root,
        std::shared_ptr<SymbolRepository> symbol_repository)
{
    // Step 1. Propagate the polarity of the root to its immediate child.
    if (sentence_root->is_negative_polarity()) {
        auto borrowed = sentence_root->take_sentence();
        borrowed->flip_polarity();
        sentence_root->put_sentence(std::move(borrowed));
        sentence_root->flip_polarity();
    }

    /*
     * Step 2. Push through the seven-stage CNF normalisation pipeline.
     *
     * This produces a MutableSentenceRoot that is structured as a tree, but only contains elements allowable
```

```
     * in a CNF tree. Note that a new tree is not created; the original tree is mutated such that it can be
     * trivially converted to the conjunctive-disjunctive set form.
     *
     * If the CNF logger is configured to an info level, the original ("before") and normalised ("after") CNF
     * statements are serialised with the TextSerialiserVisitor.
     */

    if (cnf_logger->isInfoEnabled()) {

        cnf_logger->info("Beginning CNF pipeline transformation.");
        sentence_root->accept(serialiser_visitor);
        cnf_logger->info("Initial sentence: " + serialiser_visitor.extract());

        sentence_root = cnf_normalise(std::move(sentence_root));

        cnf_logger->info("Completed CNF transformation.");
        sentence_root->accept(serialiser_visitor);
        cnf_logger->info("Normalised sentence: " + serialiser_visitor.extract());
    } else
        sentence_root = cnf_normalise(std::move(sentence_root));

    /*
     * Step 3. Populate the symbol repository.
     *
     * This transforms the normalised mutable CNF tree into the corresponding immutable form, represented by a
     * SentenceRoot. SentenceRoot objects do not indicate trees, rather sets of literals under disjunction, of
     * which the elements are under conjunction. The given SymbolRepository is also populated with the terms
     * and literals appearing in the normalised expression.
     */

    return build_symbol_repository(std::move(sentence_root), std::move(symbol_repository));
}

std::unique_ptr<MutableSentenceRoot> ExpressionFactory::cnf_normalise(
        std::unique_ptr<MutableSentenceRoot> &&denormalised_root)
{
    // clang-format off
    const std::array<std::unique_ptr<MutatingSentenceVisitorBase>, 7> cnf_visitors{
        std::make_unique<ImplicationEliminationVisitor>(),
        std::make_unique<DMLVisitor>(),
        std::make_unique<SymbolStandardisingVisitor>(),
        std::make_unique<QuantifierExtractingVisitor>(),
        std::make_unique<SkolemIntroducingVisitor>(),
        std::make_unique<UniversalEliminationVisitor>(),
        std::make_unique<DisjunctionDistributionVisitor>()
    };
    // clang-format on

    if (cnf_logger->isDebugEnabled())
        /*
         * Explicitly check if debugging is enabled on the CNF logger, as running a serialisation visitor down
         * the entire tree for each step in the normalisation pipeline would be a great inefficiency if the
         * strings were not used!
         */
        for (const auto &visitor: cnf_visitors) {
            try {
                denormalised_root->accept(*visitor);
            } catch (const SemanticException &semantic_exception) {
                Logging::get_logger({cnf_logger->getName(), std::string(visitor->get_visitor_name())})
                        ->error(semantic_exception.what());
                throw;
            }

            // Serialise result of normalising stage.
            denormalised_root->accept(serialiser_visitor);
            Logging::get_logger({cnf_logger->getName(), std::string(visitor->get_visitor_name())})
                    ->debug(serialiser_visitor.extract());
        }
    else
        for (const auto &visitor: cnf_visitors)
            try {
                denormalised_root->accept(*visitor);
            } catch (const SemanticException &semantic_exception) {
                Logging::get_logger({cnf_logger->getName(), std::string(visitor->get_visitor_name())})
                        ->error(semantic_exception.what());
                throw;
            }

    return denormalised_root;
}
```

```
std::unique_ptr<SentenceRoot> ExpressionFactory::build_symbol_repository(
        std::unique_ptr<MutableSentenceRoot> &&normalised_root,
        std::shared_ptr<SymbolRepository> symbol_repository)
{
    RepositoryBuildingVisitor building_visitor(std::move(symbol_repository));
    normalised_root->accept(building_visitor);
    return building_visitor.take_last_root();
}

} // namespace optifol
```

1.9.2 ExpressionFactory.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the CNF expression factory
 * @author Oliver Dixon
 * @date 2025-09-14
 * @version Development
 */

#ifndef OPTIFOL_EXPRESSIONFACTORY_HPP
#define OPTIFOL_EXPRESSIONFACTORY_HPP

#include <log4cxx/logger.h>

#include "../Visitors/MutableTargets/Observers/TextSerialiserVisitor.hpp"

namespace optifol
{

class MutableSentenceRoot;
class SentenceRoot;
class SymbolRepository;

/**
 * @class ExpressionFactory
 * @brief The ExpressionFactory provides a single-method static factory for normalising FOL sentences produced
 * by the FOLParser and registering the symbols in a shared-ownership SymbolRepository.
 */
class ExpressionFactory
{
public:
    /**
     * @brief Produce an immutable, CNF-normalised SentenceRoot from the given de-normalised
     * MutableSentenceRoot.
     * @param sentence_root The de-normalised MutableSentenceRoot, typically provided by a FOLParser.
     * @param symbol_repository The shared-ownership SymbolRepository into which expression symbols should be
     *  registered.
     * @return The normalised SentenceRoot.
     * @throws SemanticException A semantic/logical error was encountered with the expression during
     * normalisation or standardisation.
     */
    static std::unique_ptr<SentenceRoot> build_sentence(std::unique_ptr<MutableSentenceRoot> &&sentence_root,
            std::shared_ptr<SymbolRepository> symbol_repository);

private:
    /**
     * @brief Push an arbitrary-form MutableSentenceRoot through the seven-stage CNF normalisation pipeline,
     * but do not transform into the immutable representation or register any symbols in a SymbolRepository.
     * @param denormalised_root The de-normalised sentence root to normalise into CNF.
     * @return The MutableSentenceRoot of the CNF-normalised tree.
     * @throws SemanticException A semantic/logical error was encountered with the expression during
     * normalisation.
     */
    static std::unique_ptr<MutableSentenceRoot> cnf_normalise(
            std::unique_ptr<MutableSentenceRoot> &&denormalised_root);

    /**
     * @brief Build an immutable SentenceRoot from the given CNF-normalised MutableSentenceRoot, and register
     * symbols in the shared-ownership SymbolRepository.
     * @param normalised_root The MutableSentenceRoot of the CNF sentence.
     * @param symbol_repository The shared-ownership SymbolRepository into which expression symbols should be
     *  registered.
```

```
     * @return The normalised SentenceRoot.
     * @throws SemanticException A semantic/logical error was encountered with the expression during
     * standardisation.
     */
    static std::unique_ptr<SentenceRoot> build_symbol_repository(
            std::unique_ptr<MutableSentenceRoot> &&normalised_root,
            std::shared_ptr<SymbolRepository> symbol_repository);

    static const log4cxx::LoggerPtr cnf_logger;
    static TextSerialiserVisitor serialiser_visitor;
};

} // namespace optifol

#endif // OPTIFOL_EXPRESSIONFACTORY_HPP
```

1.9.3 Feature.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification and implementation for the Clause feature
 * @author Oliver Dixon
 * @date 2026-02-04
 * @version Development
 */

#ifndef OPTIFOL_FEATURE_HPP
#define OPTIFOL_FEATURE_HPP

#include <cassert>
#include <utility>

#include "../IHashable.hpp"

namespace optifol
{

/**
 * @class Feature
 * @brief A Feature is a metric of a Clause. It has a type and a magnitude, indicating the extent of the
 * feature type's representation within the Clause.
 */
class Feature : public IHashable
{
public:
    /**
     * @class FeatureType
     * @brief The four Clause feature types.
     */
    enum class FeatureType
    {
        MaxDepth,
        LiteralCount,
        FunctionCount,
        VariableCount
    };

    static constexpr std::array<FeatureType, 4> feature_types{FeatureType::MaxDepth,
            FeatureType::LiteralCount, FeatureType::FunctionCount, FeatureType::VariableCount};

    /**
     * @brief Construct a new feature type with zero magnitude.
     * @param feature_type The feature type to construct.
     */
    explicit Feature(const FeatureType feature_type) :
        feature_type(feature_type)
    {
    }

    /**
     * @brief Construct a new feature type with a specified magnitude.
     * @param feature_type The feature type to construct.
     * @param magnitude The initial magnitude of the feature.
     */
```

```
Feature(const FeatureType feature_type, const unsigned int magnitude) :
    feature_type(feature_type),
    magnitude(magnitude)
{
}

[[nodiscard]] std::size_t hash() const noexcept override
{
    return hash_combine(
            std::hash<std::underlying_type_t<FeatureType>>{}(std::to_underlying(feature_type)),
            std::hash<unsigned int>{}(magnitude));
}

/**
 * @brief Determine the feature's importance w.r.t. Resolution Potential.
 * @param other_feature The other feature to compare against.
 * @return Does this feature have a strictly lower RP than the other one?
 */
[[nodiscard]] bool operator<(const Feature &other_feature) const noexcept
{
    if (feature_type != other_feature.feature_type)
        return feature_type < other_feature.feature_type;

    return magnitude < other_feature.magnitude;
}

/**
 * @brief Determine the feature's importance w.r.t. Resolution Potential.
 * @param other_feature The other feature to compare against.
 * @return Does this feature have a strictly greater RP than the other one?
 */
[[nodiscard]] bool operator>(const Feature &other_feature) const noexcept
{
    if (feature_type != other_feature.feature_type)
        return feature_type > other_feature.feature_type;

    return magnitude > other_feature.magnitude;
}

/**
 * @brief Reset the feature's magnitude back to its initial value.
 */
void reset() noexcept
{
    magnitude = 0;
}

/**
 * @brief Increment the feature by a given amount
 * @param amount The extent by which the feature's magnitude should be incremented (default 1).
 */
void increment(const unsigned int amount = 1) noexcept
{
    magnitude += amount;
}

/**
 * @brief Maximise the feature with another magnitude.
 * @param candidate The candidate magnitude to maximise.
 */
void maximise(const unsigned int candidate) noexcept
{
    magnitude = std::max(magnitude, candidate);
}

/**
 * @brief Combine the given feature with this one.
 * @param new_feature The other feature with which to combine.
 * @pre The given feature must be of the same type.
 */
void join(const Feature &new_feature) noexcept
{
    assert(feature_type == new_feature.feature_type);

    switch (feature_type) {
    case FeatureType::MaxDepth:
        maximise(new_feature.magnitude);
        break;
    case FeatureType::FunctionCount:
    case FeatureType::LiteralCount:
    case FeatureType::VariableCount:
```

```
            increment(new_feature.magnitude);
            break;
        }
    }

    [[nodiscard]] unsigned int get_magnitude() const noexcept
    {
        return magnitude;
    }

    [[nodiscard]] FeatureType get_feature_type() const noexcept
    {
        return feature_type;
    }

    /**
     * @brief Static helper to retrieve a feature of the specified type from a list of features.
     * @param features The list of features.
     * @param feature_type The feature type of retrieve.
     * @return The requested feature.
     */
    static Feature &get(std::vector<Feature> &features, const FeatureType feature_type)
    {
        const auto index = static_cast<std::size_t>(std::to_underlying(feature_type));
        assert(index < features.size());

        auto &candidate = features[index];
        assert(candidate.feature_type == feature_type);

        return candidate;
    }

private:
    const FeatureType feature_type;
    unsigned int magnitude = 0;
};

} // namespace optifol

#endif // OPTIFOL_FEATURE_HPP
```

1.9.4 FVIKnowledgeBase.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Feature Vector Indexing Clause knowledge base
 * @author Oliver Dixon
 * @date 2026-02-09
 * @version Development
 */

#include "FVIKnowledgeBase.hpp"

#include <ranges>

#include "../IR/Sentences/Clause.hpp"
#include "../Logging.hpp"
#include "../Visitors/RegularTargets/Unification/UnificationVisitor.hpp"

namespace optifol
{

const log4cxx::LoggerPtr FVIKnowledgeBase::kb_logger =
        Logging::get_logger({"LogicServices", "FVIKnowledgeBase"});

FVIKnowledgeBase::FVIKnowledgeBase(std::shared_ptr<SymbolRepository> symbol_repository) :
    symbol_repository(std::move(symbol_repository)),
    unification_visitor(this->symbol_repository)
{
}

std::pair<UniqueUnorderedSet<Clause>::iterator, bool> FVIKnowledgeBase::insert_clause(
        std::unique_ptr<Clause> &&clause,
        const std::optional<std::function<void(std::unique_ptr<Clause> &&)>> &rejection_handler)
```

```
{
    FVINode *node = &root;
    const auto &clause_features = clause->observe_features();

    for (const auto &feature: clause_features) {
        auto &child = node->children[feature];
        if (child == nullptr)
            child = std::make_unique<FVINode>();
        node = child.get();
    }

    if (node->clause_set.contains(clause)) {
        LOG4CXX_INFO(kb_logger,
                std::format("Rejecting clause {} because it already exists in the KB trie.", *clause));
        if (rejection_handler.has_value())
            (*rejection_handler)(std::move(clause));
        return {node->clause_set.end(), false};
    }

    LOG4CXX_INFO(kb_logger, std::format("Accepting clause {} into the KB.", *clause));
    auto [it, success] = node->clause_set.insert(std::move(clause));
    assert(success);
    ++total_clause_count;
    LOG4CXX_DEBUG(kb_logger, std::format("The KB now contains {} clauses.", total_clause_count));
    return {it, success};
}

std::pair<UniqueUnorderedSet<Clause>::iterator, bool> FVIKnowledgeBase::add_clause(
        std::unique_ptr<Clause> &&clause,
        const std::optional<std::function<void(std::unique_ptr<Clause> &&)>> &rejection_handler)
{
    std::vector<const Clause *> subsuming_clauses;
    get_subsuming(*clause, root, 0, subsuming_clauses);

    if (!subsuming_clauses.empty()) {
        LOG4CXX_INFO(kb_logger,
                std::format("Rejecting clause {} because it would be subsumed by current KB.", *clause));
        if (rejection_handler.has_value())
            (*rejection_handler)(std::move(clause));
        return {root.clause_set.end(), false};
    }

    remove_subsumed(*clause, root, 0);
    return insert_clause(std::move(clause), rejection_handler);
}

std::generator<const Clause *> FVIKnowledgeBase::flatten() const
{
    std::vector<const FVINode *> stack;
    stack.push_back(&root);

    while (!stack.empty()) {
        const auto node = stack.back();
        stack.pop_back();

        for (const auto &clause: node->clause_set)
            co_yield clause.get();

        for (const auto &[_, child_node]: node->children)
            stack.push_back(child_node.get());
    }
}

void FVIKnowledgeBase::get_subsuming(const Clause &clause, const FVINode &node, const unsigned int depth,
        std::vector<const Clause *> &subsuming_clauses) const
{
    const auto &features = clause.observe_features();
    if (depth >= features.size())

        // The given node is a leaf node.
        for (const auto contained_clause: node.clause_set | unwrap_clause) {
            const auto subsumes = contained_clause->subsumes(clause, unification_visitor);
            unification_visitor.reset_substitutions();
            if (subsumes)
                subsuming_clauses.push_back(contained_clause);
            else
                return;
        }

    else {
```

```
        // The given node is not a leaf node.
        const auto &feature = features[depth];
        const auto filter_unrelated_features = std::ranges::views::drop_while(
                [&feature](const auto &candidate_pair) { return candidate_pair.first < feature; });

        const auto select_lesser_magnitudes = std::ranges::views::take_while(
                [&feature](const auto &candidate_pair)
                {
                    return candidate_pair.first.get_feature_type() == feature.get_feature_type() &&
                            candidate_pair.first.get_magnitude() <= feature.get_magnitude();
                });

        for (const auto &[child_feature, child_node]:
                node.children | filter_unrelated_features | select_lesser_magnitudes)
            get_subsuming(clause, *child_node, depth + 1, subsuming_clauses);

        get_subsuming(clause, node, depth + 1, subsuming_clauses);
    }
}

void FVIKnowledgeBase::get_subsumed(const Clause &clause, const FVINode &node, const unsigned int depth,
        std::vector<const Clause *> &subsumed_clauses) const
{
    const auto &features = clause.observe_features();

    if (depth >= features.size())

        // The given node is a leaf node.
        explore_leaf(clause, node, subsumed_clauses);

    else {

        // The given node is not a leaf node.
        const auto filter_greater_magnitudes = std::ranges::views::take_while(
                [depth, &features](const auto &candidate_pair)
                {
                    if (depth == 0)
                        return true;

                    return candidate_pair.first > features[depth - 1];
                });

        const auto &current_feature = features[depth];
        for (const auto &[child_feature, child_node]:
                node.children | std::ranges::views::reverse | filter_greater_magnitudes)
            if (child_feature.get_feature_type() <= current_feature.get_feature_type()) {
                const std::size_t offset =
                        child_feature.get_feature_type() == current_feature.get_feature_type() &&
                                child_feature.get_magnitude() >= current_feature.get_magnitude()
                        ? 1
                        : 0;

                get_subsumed(clause, *child_node, depth + offset, subsumed_clauses);
            }
    }
}

void FVIKnowledgeBase::explore_leaf(
        const Clause &clause, const FVINode &node, std::vector<const Clause *> &subsumed_clauses) const
{
    for (const auto contained_clause: node.clause_set | unwrap_clause) {
        const auto subsumes = clause.subsumes(*contained_clause, unification_visitor);
        unification_visitor.reset_substitutions();
        if (subsumes)
            subsumed_clauses.push_back(contained_clause);
    }

    for (const auto &[feature, child_node]: node.children)
        explore_leaf(clause, *child_node, subsumed_clauses);
}

void FVIKnowledgeBase::remove_subsumed(const Clause &clause, FVINode &node, unsigned int depth)
{
    const auto &features = clause.observe_features();

    if (depth >= features.size())

        // The given node is a leaf node.
        explore_and_remove_leaf(clause, node);

    else {
```

```
        // The given node is not a leaf node.
        const auto filter_greater_magnitudes = std::ranges::views::take_while(
                [depth, &features](const auto &candidate_pair)
                {
                    if (depth == 0)
                        return true;

                    return candidate_pair.first > features[depth - 1];
                });

        const auto &current_feature = features[depth];
        std::vector<Feature> slated_for_removal;

        for (const auto &[child_feature, child_node]:
                node.children | std::ranges::views::reverse | filter_greater_magnitudes)

            if (child_feature.get_feature_type() <= current_feature.get_feature_type()) {
                const std::size_t offset =
                        child_feature.get_feature_type() == current_feature.get_feature_type() &&
                                child_feature.get_magnitude() >= current_feature.get_magnitude()
                        ? 1
                        : 0;

                remove_subsumed(clause, *child_node, depth + offset);
                if (child_node->children.empty() && child_node->clause_set.empty())
                    slated_for_removal.push_back(child_feature);
            }

        for (const auto &target: slated_for_removal)
            node.children.erase(target);
    }
}

void FVIKnowledgeBase::explore_and_remove_leaf(const Clause &incoming_clause, FVINode &node)
{
    auto next_clause_it = node.clause_set.begin();
    for (auto clause_it = next_clause_it; clause_it != node.clause_set.end(); clause_it = next_clause_it) {
        ++next_clause_it;
        const auto subsumes = incoming_clause.subsumes(**clause_it, unification_visitor);
        unification_visitor.reset_substitutions();
        if (subsumes) {
            LOG4CXX_INFO(kb_logger,
                    std::format("Orphaning clause {} from the KB as it is being subsumed by {}.", **clause_it,
                            incoming_clause));

            orphaned_clauses.insert(std::move(node.clause_set.extract(clause_it).value()));
            --total_clause_count;

            LOG4CXX_DEBUG(kb_logger,
                    std::format("The KB now contains {} active and {} orphaned clauses.", total_clause_count,
                            orphaned_clauses.size()));
        }
    }

    std::vector<Feature> slated_for_removal;
    for (const auto &[child_feature, child_node]: node.children) {
        explore_and_remove_leaf(incoming_clause, *child_node);
        if (child_node->children.empty() && child_node->clause_set.empty())
            slated_for_removal.push_back(child_feature);
    }

    for (const auto &target: slated_for_removal)
        node.children.erase(target);
}

unsigned int FVIKnowledgeBase::get_total_clause_count() const noexcept
{
    return total_clause_count;
}

bool FVIKnowledgeBase::is_orphaned(const Clause *clause) const
{
    return orphaned_clauses.contains(clause);
}

} // namespace optifol
```

1.9.5 FVIKnowledgeBase.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Feature Vector Indexing Clause knowledge base
 * @author Oliver Dixon
 * @date 2026-02-09
 * @version Development
 */

#ifndef OPTIFOL_FVIKNOWLEDGEBASE_HPP
#define OPTIFOL_FVIKNOWLEDGEBASE_HPP

#include <functional>
#include <generator>
#include <log4cxx/logger.h>
#include <map>
#include <memory>
#include <vector>

#include "../IR/Sentences/Clause.hpp"
#include "../Optifol.hpp"
#include "../Visitors/RegularTargets/Unification/UnificationVisitor.hpp"
#include "Feature.hpp"

namespace optifol
{

class SymbolRepository;

/**
 * @class FVIKnowledgeBase
 * @brief The FVI (feature vector-indexed) KB stores clauses in a KB-like structure i.a.w. their features. See
 * the Schulz paper for more information on its nature and supported operations.
 */
class FVIKnowledgeBase
{
public:
    explicit FVIKnowledgeBase(std::shared_ptr<SymbolRepository> symbol_repository);

    /**
     * @brief Introduces (and transfers ownership of) a new Clause into the knowledge base.
     * @details This function introduces a Clause to the KB on the condition that it is not subsumed by the
     * existing KB. Furthermore, if its introduction causes clauses to be subsumed, those latter clauses are
     * removed.
     * @param clause The Clause to introduce.
     * @param rejection_handler The callback to which ownership should be handed back if the knowledge base
     * refuses the clause; this happens when the clause would be immediately subsumed by the KB.
     * @return An iterator to the inserted clause, or an <code>end</code> if refused, and a boolean flag
     * indicating acceptance of the Clause by the KB.
     */
    std::pair<UniqueUnorderedSet<Clause>::iterator, bool> add_clause(std::unique_ptr<Clause> &&clause,
            const std::optional<std::function<void(std::unique_ptr<Clause> &&)>> &rejection_handler = {});

    [[nodiscard]] std::generator<const Clause *> flatten() const;

    [[nodiscard]] unsigned int get_total_clause_count() const noexcept;

    [[nodiscard]] bool is_orphaned(const Clause *clause) const;

private:
    struct FVINode
    {
        FVINode() = default;

        std::map<Feature, std::unique_ptr<FVINode>> children;
        UniqueUnorderedSet<Clause> clause_set;
    };

    struct OrphanHasher
    {
        using is_transparent = void;

        [[nodiscard]] std::size_t operator()(const Clause *const ptr) const noexcept
        {
```

```
            return reinterpret_cast<std::size_t>(ptr);
        }

        [[nodiscard]] std::size_t operator()(const std::unique_ptr<Clause> &unq) const noexcept
        {
            return reinterpret_cast<std::size_t>(unq.get());
        }
    };

    struct OrphanEquality
    {
        using is_transparent = void;

        [[nodiscard]] bool operator()(const Clause *const lhs_ptr, const Clause *const rhs_ptr) const noexcept
        {
            return lhs_ptr == rhs_ptr;
        }

        [[nodiscard]] bool operator()(
                const Clause *const lhs_ptr, const std::unique_ptr<Clause> &rhs_unq) const noexcept
        {
            return lhs_ptr == rhs_unq.get();
        }

        [[nodiscard]] bool operator()(
                const std::unique_ptr<Clause> &lhs_unq, const Clause *const rhs_ptr) const noexcept
        {
            return lhs_unq.get() == rhs_ptr;
        }

        [[nodiscard]] bool operator()(
                const std::unique_ptr<Clause> &lhs_unq, const std::unique_ptr<Clause> &rhs_unq) const noexcept
        {
            return lhs_unq.get() == rhs_unq.get();
        }
    };

    static constexpr auto unwrap_clause =
            std::views::transform([](const std::unique_ptr<Clause> &clause) { return clause.get(); });

    static const log4cxx::LoggerPtr kb_logger;

    std::pair<UniqueUnorderedSet<Clause>::iterator, bool> insert_clause(std::unique_ptr<Clause> &&clause,
            const std::optional<std::function<void(std::unique_ptr<Clause> &&)>> &rejection_handler);

    void get_subsuming(const Clause &clause, const FVINode &node, unsigned int depth,
            std::vector<const Clause *> &subsuming_clauses) const;

    void get_subsumed(const Clause &clause, const FVINode &node, unsigned int depth,
            std::vector<const Clause *> &subsumed_clauses) const;

    void explore_leaf(
            const Clause &clause, const FVINode &node, std::vector<const Clause *> &subsumed_clauses) const;

    void remove_subsumed(const Clause &clause, FVINode &node, unsigned int depth);

    void explore_and_remove_leaf(const Clause &incoming_clause, FVINode &node);

    FVINode root;
    std::shared_ptr<SymbolRepository> symbol_repository;
    unsigned int total_clause_count = 0;

    std::unordered_set<std::unique_ptr<Clause>, OrphanHasher, OrphanEquality> orphaned_clauses;
    mutable UnificationVisitor unification_visitor;
};

} // namespace optifol

#endif // OPTIFOL_FVIKNOWLEDGEBASE_HPP
```

1.9.6 ProofTreeNode.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
```

```
 * @brief Class specification and implementation for the Proof Tree Node base class
 * @author Oliver Dixon
 * @date 2026-01-29
 * @version Development
 */

#ifndef OPTIFOL_PROOFTREENODE_HPP
#define OPTIFOL_PROOFTREENODE_HPP

#include <memory>

#include "../ISerialisable.hpp"

namespace optifol
{

struct Unifier;
class Clause;
class Resolvent;

/**
 * @class ProofTreeNode
 * @brief A Proof Tree Node is a graphical object that may appear within a deduction trace, or, a Minimally
 * Spanning Deduction Graph. They consist of zero or two parents, an optional unifier clause, and a resolvent
 * or axiom node.
 */
class ProofTreeNode : public ISerialisable
{
public:
    ProofTreeNode(const ProofTreeNode &) = default;
    ProofTreeNode &operator=(const ProofTreeNode &) = default;

    [[nodiscard]] const ProofTreeNode *observe_lhs_parent() const noexcept
    {
        return lhs_parent;
    }

    [[nodiscard]] const ProofTreeNode *observe_rhs_parent() const noexcept
    {
        return rhs_parent;
    }

    [[nodiscard]] virtual const Clause *observe_node() const noexcept = 0;

    [[nodiscard]] virtual std::optional<const Unifier *> observe_edge() const noexcept = 0;

    [[nodiscard]] unsigned int get_depth() const noexcept
    {
        return depth;
    }

    [[nodiscard]] virtual bool is_unit() const noexcept = 0;

protected:
    ProofTreeNode() = default;

    ProofTreeNode(const ProofTreeNode *const lhs_parent, const ProofTreeNode *const rhs_parent) :
        lhs_parent(lhs_parent),
        rhs_parent(rhs_parent),
        depth(std::max(lhs_parent->get_depth(), rhs_parent->get_depth()) + 1)
    {
    }

private:
    const ProofTreeNode *lhs_parent = nullptr;
    const ProofTreeNode *rhs_parent = nullptr;
    unsigned int depth = 0;
};

} // namespace optifol

#endif // OPTIFOL_PROOFTREENODE_HPP
```

1.9.7 Prover.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */
```

```cpp
/**
 * @file
 * @brief Class implementation for the FOL Resolution Knowledge Base
 * @author Oliver Dixon
 * @date 2025-09-15
 * @version Development
 */

#include "Prover.hpp"

#include <algorithm>
#include <queue>
#include <ranges>
#include <sigc++/adaptors/bind.h>

#include "../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"
#include "../IR/Sentences/Literal.hpp"
#include "../IR/SymbolRepository.hpp"
#include "../Logging.hpp"
#include "ExpressionFactory.hpp"
#include "QueryResult.hpp"
#include "Resolvent.hpp"

namespace optifol
{

const log4cxx::LoggerPtr Prover::prover_logger = Logging::get_logger({"LogicServices", "Prover"});
const log4cxx::LoggerPtr Prover::resolution_logger =
        Logging::get_logger({"LogicServices", "Prover", "Resolution"});
const log4cxx::LoggerPtr Prover::factoring_logger =
        Logging::get_logger({"LogicServices", "Prover", "Factoring"});

Prover::Prover(std::shared_ptr<SymbolRepository> symbol_repository) :
    symbol_repository(std::move(symbol_repository)),
    base_clauses(this->symbol_repository),
    unifier(this->symbol_repository),
    factoring_unifier(this->symbol_repository),
    applicator(unifier.share_substitutions(), this->symbol_repository),
    factoring_applicator(factoring_unifier.share_substitutions(), this->symbol_repository)
{
}

bool Prover::tell(const SentenceRoot &sentence)
{
    return std::ranges::all_of(sentence, [this](const Clause &new_clause) { return tell(new_clause); });
}

bool Prover::tell(const Clause &new_clause)
{
    return base_clauses.add_clause(std::make_unique<Clause>(new_clause)).second;
}

bool Prover::PQResolventUnitPref::operator()(
        const std::unique_ptr<Resolvent> &lhs, const std::unique_ptr<Resolvent> &rhs) noexcept
{
    return *lhs < *rhs;
}

QueryResult Prover::run_resolution(std::unique_ptr<SentenceRoot> &&negated_query, const size_t max_step_count)
{
    QueryResult result(
            symbol_repository); // Execution trace, introduced clauses, and metadata built up by the solver.

    // Introduce the negated goal clauses into the query instance.
    for (const auto &clause: *negated_query)
        result.introduced_clauses.add_clause(std::make_unique<Clause>(clause));

    /*
     * Over the Cartesian product of clauses in the KB, search for resolvents in the initial set and populate
     * the working queue accordingly. Note that binary resolution is a commutative operation, so we restrict
     * the RHS.
     */

    ResolventQueue working_queue; // Generated resolvents not yet committed to introduced knowledge.
    for (const auto lhs_clause:
            std::ranges::concat_view(base_clauses.flatten(), result.introduced_clauses.flatten()))
        for (const auto &rhs_clause:
                std::ranges::concat_view(base_clauses.flatten(), result.introduced_clauses.flatten())) {
            auto new_resolvents = find_resolvents(lhs_clause, rhs_clause);
            for (auto &&[resolvent, owning_resolution]: new_resolvents)
```

```
            working_queue.push(std::move(resolvent), std::move(owning_resolution));
    }

resolution_logger->debug(
        std::format("Resolution initial search gathered {} resolvents.", working_queue.size()));
RawUnorderedSet<const Clause>
        seen_resolvents; // Transparent hashing to provide lightweight de-duplication.

/*
 * Once the initial set of resolvents has been added to the priority queue, continue stepping until a
 * result has been produced to indicate whether the negated query induces an inconsistent KB.
 */
for (; !working_queue.empty(); ++result.elapsed_step_count) {
    auto next_resolvent = working_queue.pop();

    /*
     * Mark the resolved clause as "seen". If it is being seen for the first time, the popped resolvent is
     * committed to the final execution trace, and ownership of the resolved clause is transferred
     * likewise. If it has been seen before, we can just ignore it, as binary resolution can repeat
     * resolutions for different clauses.
     */
    auto [_, was_unseen] = seen_resolvents.insert(next_resolvent.observe_node());

    if (was_unseen) {
        resolution_logger->info(std::format("Step {} is using resolution {}.", result.elapsed_step_count,
                *next_resolvent.observe_node()));

        if (result.introduced_clauses.is_orphaned(next_resolvent.observe_node()))
            // Keep the (non-owning) resolvent popped off the working queue, but don't move its
            // resolution.
            continue;

        /*
         * Insert the next-unseen resolvent into the KB by transferring ownership of the corresponding
         * resolution into the QueryResult and recording the resolvent into the trace. If the resolution
         * clause is non-trivial and new, search for more resolvents.
         *
         * If the KB refuses ownership of the new Clause (i.e. if it would be subsumed by the existing
         * KB), we take ownership back and return it to the working queue. Otherwise, it would go out of
         * scope here despite other resolvents potentially still using it.
         */
        auto owning_resolution = working_queue.extract_resolution(next_resolvent);

        auto [committed_resolution_it, was_committed] =
                result.introduced_clauses.add_clause(std::move(owning_resolution),
                        sigc::mem_fun(working_queue, &ResolventQueue::store_resolution));
        result.relations.push_back(std::move(next_resolvent));

        if ((*committed_resolution_it)->get_triviality_state() == Clause::State::TriviallyFalse) {
            // An empty derived clause indicates that a contradiction was derived in the KB.
            result.outcome = QueryResult::ConjectureStatus::Consistent;
            result.terminating_resolvent = &result.relations.back();
            working_queue.dump(result.relations, result.introduced_clauses);
            return result;
        }

        if (was_committed) {
            const Resolvent *const lhs = &result.relations.back();

            // Clauses on RHS
            for (const auto rhs_clause: base_clauses.flatten()) {
                auto new_resolvents = find_resolvents(lhs, rhs_clause);
                for (auto &&[new_resolvent, new_resolution]: new_resolvents)
                    working_queue.push(std::move(new_resolvent), std::move(new_resolution));
            }

            // Resolvents on RHS
            for (const auto &rhs_resolvent: result.relations) {
                auto new_resolvents = find_resolvents(lhs, &rhs_resolvent);
                for (auto &&[new_resolvent, new_resolution]: new_resolvents)
                    working_queue.push(std::move(new_resolvent), std::move(new_resolution));
            }
        }
    } else
        resolution_logger->debug(std::format(
                "Resolution skipping step {} due to repeated resolvent.", result.elapsed_step_count));

    if (result.elapsed_step_count == max_step_count) {
        result.outcome = QueryResult::ConjectureStatus::TimedOut;
        working_queue.dump(result.relations, result.introduced_clauses);
        return result;
```

```
        }
    }

    /*
     * If we have exhausted all resolvents in the PQ without finding a contradiction, then introduced of the
     * negated query did not induce an inconsistent KB.
     */
    result.outcome = QueryResult::ConjectureStatus::Inconsistent;
    working_queue.dump(result.relations, result.introduced_clauses);
    return result;
}

QueryResult Prover::ask(std::unique_ptr<MutableSentenceRoot> &&query, const size_t max_step_count)
{
    // Produce the negation of the goal.
    query->flip_polarity();
    auto negated_query = ExpressionFactory::build_sentence(std::move(query), symbol_repository);

    const auto clause_count = base_clauses.get_total_clause_count();
    prover_logger->info(std::format(
            "Querying the KB of {} clauses for the negation of {}.", clause_count, *negated_query));

    if (prover_logger->isTraceEnabled()) {
        prover_logger->trace("Dumping initial knowledge base...");
        unsigned int clause_idx = 1;
        for (const auto clause: base_clauses.flatten())
            prover_logger->trace(std::format("KB Clause {}/{}: {}", clause_idx++, clause_count, *clause));

        prover_logger->trace(std::format("KB Clause (negated goal): {}", *negated_query->begin()));
    }

    // Run the theorem-prover.
    QueryResult result = run_resolution(std::move(negated_query), max_step_count);

    switch (result.outcome) {
    case QueryResult::ConjectureStatus::TimedOut:
        prover_logger->info(std::format("The solver timed out after {} steps.", max_step_count));
        break;
    case QueryResult::ConjectureStatus::Consistent:
        prover_logger->info(
                std::format("A contradiction was derived in {} steps; the queried sentence is consistent.",
                        result.elapsed_step_count));
        break;
    case QueryResult::ConjectureStatus::Inconsistent:
        prover_logger->info(std::format(
                "A contradiction could not be derived after {} steps, and no more resolutions were "
                "available. The queried sentence is inconsistent.",
                result.elapsed_step_count));
        break;
    case QueryResult::ConjectureStatus::NotExecuted:
        prover_logger->error("The solver panicked, and no proof was attempted.");
        break;
    }

    return result;
}

std::unique_ptr<Clause> Prover::collect_unified_literals(
        const Literal &self, const Clause &source_clause, std::unique_ptr<Clause> &&destination_clause) const
{
    const auto transformer =
            std::views::filter([self](const Literal *candidate) { return !self.operator==(*candidate); }) |
            std::views::transform([this](const Literal *target) { return target->accept(applicator); });

    for (const auto transformed_literal: source_clause | transformer)
        destination_clause->add_literal(transformed_literal);

    return destination_clause;
}

std::unique_ptr<Clause> Prover::factor_literals(std::unique_ptr<Clause> &&unified_clause)
{
    bool factoring_done = false;
    bool trivially_true = false;
    auto working_clause = std::make_unique<Clause>(*unified_clause);

    do {
        factoring_done = false;

        for (const auto [factoring_idx, factoring_lhs_literal]: *unified_clause | std::views::enumerate) {
            for (const auto factoring_rhs_literal: *unified_clause | std::views::take(factoring_idx)) {
```

```
                if (working_clause != nullptr &&
                        working_clause->get_triviality_state() == Clause::State::TriviallyTrue) {
                    trivially_true = true;
                    break;
                }

                /*
                 * Finding a unifying MGU between the factoring literals indicates an opportunity to reduce
                 * the clause size. Once the MGU is obtained, we can replace the working clause with the
                 * simplified variant, under application of the MGU.
                 */
                if (factoring_lhs_literal->accept(factoring_unifier, *factoring_rhs_literal)) {
                    factoring_logger->info(std::format("Successfully unified {} and {}.",
                            *factoring_lhs_literal, *factoring_rhs_literal));

                    auto simplified = std::make_unique<Clause>();
                    std::ranges::for_each(*unified_clause, [this, &simplified](const Literal *literal)
                            { simplified->add_literal(literal->accept(factoring_applicator)); });

                    factoring_logger->info(std::format(
                            "Factoring produced a new working clause of order {}.", simplified->order()));

                    working_clause.swap(simplified);
                    factoring_applicator.keep_new_symbols();
                    factoring_done = true;
                    break;
                }
            }

            if (!trivially_true || factoring_done)
                break;
        }
    } while (!trivially_true && factoring_done);

    return working_clause;
}

bool Prover::insert_clause(std::unique_ptr<Clause> &&new_clause, UniqueUnorderedSet<Clause> &destination)
{
    const auto [node_it, added_ok] = destination.emplace(std::move(new_clause));
    std::ignore = node_it;

    if (!added_ok) {
        prover_logger->info("Rejecting clause from the KB as it is already present.");
        return false;
    }

    return true;
}

std::vector<std::pair<Resolvent, std::unique_ptr<Clause>>> Prover::find_resolvents(
        const ProofTreeNode *const lhs_node, const ProofTreeNode *const rhs_node)
{
    std::vector<std::pair<Resolvent, std::unique_ptr<Clause>>> resolvents;

    const auto *const lhs_clause = lhs_node->observe_node();
    const auto *const rhs_clause = rhs_node->observe_node();

    for (const auto lhs_literal: *lhs_clause)
        for (const auto rhs_literal: *rhs_clause) {

            // Attempt to unify the LHS literal with the negation of the RHS literal.
            auto rhs_args_copy = rhs_literal->observe_arguments();
            const Literal negated_rhs(std::string(rhs_literal->get_name()), std::move(rhs_args_copy),
                    rhs_literal->is_negative_polarity());

            if (lhs_literal->accept(unifier, negated_rhs)) {

                /*
                 * If the LHS and negated RHS can be unified, we have attained a set of most-general unifiers.
                 * Construct the final resolvents according to the predicate resolution rule: apply the
                 * unifier to the union of all literals except the pair removed by unification. For the LHS
                 * and RHS clauses, over all literals except the pair removed by resolution, apply the MGU and
                 * add to the resolvent set.
                 *
                 * Note that application of the MGU might create new literals. In that case, they are added to
                 * the general symbol store so we only have to see non-owning, raw, immutable pointers.
                 */

                auto resolution = std::make_unique<Clause>();
                resolution = collect_unified_literals(*lhs_literal, *lhs_clause, std::move(resolution));
```

```
                resolution = collect_unified_literals(*rhs_literal, *rhs_clause, std::move(resolution));

                resolution_logger->debug(
                        std::format("Constructed a unified clause of {} literals.", resolution->order()));

                /*
                 * Attempt to simplify the unified clause, and guarantee a complete inference process, by:
                 *
                 *  1. Eliminating entire clauses that are trivially true, containing self-negating literals;
                 * and
                 *  2. Eliminating literals that can be unified.
                 *
                 * Once the clause has been built according to the above exclusion criteria, it is added to
                 * the resolvent vector returned to the caller.
                 *
                 * Note that (1) doesn't need explicit handling here, as the clause will reduce to
                 * TriviallyTrue if a pair of complementary literals are added.
                 */

                if (resolution->get_triviality_state() == Clause::State::TriviallyTrue) {
                    resolution_logger->debug(
                            "Before factoring, the unified clause is a tautology; continuing.");
                    applicator.discard_new_symbols();
                    continue;
                }

                resolution = factor_literals(std::move(resolution));

                if (resolution->get_triviality_state() == Clause::State::TriviallyTrue) {
                    resolution_logger->debug(
                            "After factoring, the unified clause is a tautology; continuing.");
                    applicator.discard_new_symbols();
                    continue;
                }

                /*
                 * A non-trivial clause produced through resolution should be considered a fresh resolvent.
                 * This is described by the source LHS and RHS clauses, the MGU, and the unified clause of the
                 * post- application literals as a clause under disjunction.
                 */

                resolvents.emplace_back(
                        Resolvent(lhs_node, rhs_node, *unifier.observe_substitutions(), resolution.get()),
                        std::move(resolution));

                resolution_logger->debug(std::format("Resolved {} and {} to {}.", *lhs_clause, *rhs_clause,
                        *resolvents.back().first.observe_node()));

                // The applicator might have introduced new symbols, so we inherit them into the
                // SymbolRepository here.
                applicator.keep_new_symbols();
            }

            // Any important MGUs have been copied into a resolvent.
            unifier.reset_substitutions();
        }

    return resolvents;
}

} // namespace optifol
```

1.9.8 Prover.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the FOL resolution prover
 * @author Oliver Dixon
 * @date 2025-09-14
 * @version Development
 */

#ifndef OPTIFOL_PROVER_HPP
#define OPTIFOL_PROVER_HPP
```

```cpp
#include <log4cxx/logger.h>
#include <ranges>

#include "../IR/Sentences/Clause.hpp"
#include "../Visitors/RegularTargets/Unification/BidirectionalUnificationVisitor.hpp"
#include "FVIKnowledgeBase.hpp"

namespace optifol
{

class MutableSentenceRoot;
class SymbolRepository;
class Variable;
class IProcessedTerm;

class Resolvent;
struct QueryResult;

/**
 * @class Prover
 * @brief A Prover manages the storage of CNF-normalised Clause objects and allows users to query the validity
 * of statements under the specified model.
 * @details
 *  <p>
 *      The typical workflow is straightforward:
 *      <ol>
 *          <li>The user creates a new Prover instance linked to a persistent SymbolRepository;</li>
 *          <li>The user issues @ref tell commands to incrementally build the axioms of the Prover's
 * contextual knowledge base;</li> <li>Once the Prover has been informed of all the axioms, the user issues an
 * @ref ask command with a query to attempt to derive a contradiction or model proof attesting to the
 * consistency of the given conjecture under the axioms.</li> <li>After the proof has completed, the user
 * browses the execution metadata and proof trace (organised as a binary tree between Clauses and Resolvents).
 * This can be directly visualised.</li>
 *      </ol>
 *  </p>
 *  <p>
 *      Users should be careful with the data-ownership semantics of the Prover. QueryResults may observe any
 * items in the SymbolRepository, and also any base clauses (issued with @ref tell) maintained by the Prover.
 * Therefore, the Prover should outlive the SymbolRepository, which should in turn outlive the Prover. This
 * model is required due to the performance implications of copying Clauses and Resolvents.
 *  </p>
 */
class Prover
{
public:
    /**
     * @brief Create a new Prover instance to store Clause objects with reference to the shared
     * SymbolRepository.
     * @param symbol_repository The repository storing symbols used by Clause objects in the Prover.
     */
    explicit Prover(std::shared_ptr<SymbolRepository> symbol_repository);

    /**
     * @brief Attempt to introduce an entire SentenceRoot into the knowledge base.
     * @param sentence The SentenceRoot containing the Clause nodes to introduce.
     * @return Were all Clause objects from the SentenceRoot added?
     */
    bool tell(const SentenceRoot &sentence);

    /**
     * @brief Attempt to introduce a new single Clause into the knowledge base.
     * @param new_clause The new Clause to introduce.
     * @return Was the Clause added?
     */
    bool tell(const Clause &new_clause);

    /**
     * @brief Attempt to determine truth of the given query given the knowledge provided to the Prover.
     * @param query The conjecture.
     * @param max_step_count The maximum number of steps taken by the resolution procedure before issuing a
     * time-out.
     * @return The result of the query, including a ConjectureState and execution trace.
     * @note Take note of the Prover class documentation on data lifetimes before making use of the
     * QueryResult.
     */
    QueryResult ask(std::unique_ptr<MutableSentenceRoot> &&query,
            size_t max_step_count = std::numeric_limits<std::size_t>::max());

    Prover(Prover &&) = default;
```

```
private:
    /**
     * @struct PQResolventUnitPref
     * @brief Helper comparator for imposing the unit-preference strict weak ordering on owned Resolvent
     * objects.
     */
    struct PQResolventUnitPref
    {
        /**
         * @brief Is the Resolvent owned by the LHS more preferable than the Resolvent owned by the RHS?
         * @param lhs LHS Resolvent owning container
         * @param rhs RHS Resolvent owning container
         * @return LHS-RHS relation
         */
        static bool operator()(
                const std::unique_ptr<Resolvent> &lhs, const std::unique_ptr<Resolvent> &rhs) noexcept;
    };

    /**
     * @brief Drive and organise the binary resolution procedure on the populated knowledge base, attempting
     * to derive a contradiction.
     * @param negated_query A CNF-normalised construction of the negation of the goal.
     * @param max_step_count The maximum number of steps to perform in the resolution procedure before bailing
     * out.
     * @return Execution metadata and a proof trace.
     * @invariant Any Clauses referenced by Resolvents must be present in one of the standard FVIKnowledgeBase
     *  locations:
     *  <ul>
     *      <li>The <i>base clauses</i> store of the Prover; or</li>
     *      <li>The <i>introduced clauses</i> store in QueryResult.</li>
     *  </ul>
     */
    QueryResult run_resolution(std::unique_ptr<SentenceRoot> &&negated_query, size_t max_step_count);

    /**
     * @brief Apply the state UnificationApplicationVisitor to the given source Clause, filtering the given
     * Literal, and add the results to the given destination Clause.
     * @param self The Literal contained within the source Clause to filter out.
     * @param source_clause The Clause containing Literal objects to be subject to the applicator.
     * @param destination_clause The Clause to receive the filtered and transformed Literal objects from the
     * source.
     */
    std::unique_ptr<Clause> collect_unified_literals(const Literal &self, const Clause &source_clause,
            std::unique_ptr<Clause> &&destination_clause) const;

    /**
     * @brief Attempt to simplify the given Clause, ideally to a tautology, using the factoring unifier.
     * @param unified_clause The owning container of the target Clause.
     * @return The owning container of the maximally simplified (under unifier application) Clause.
     */
    std::unique_ptr<Clause> factor_literals(std::unique_ptr<Clause> &&unified_clause);

    /**
     * @brief Attempt to insert a new Clause in the given @ref std::unordered_set, providing suitable logging.
     * @param new_clause The incoming Clause to consider adding.
     * @param destination The destination set into which the Clause should be inserted.
     * @return Was the new Clause added?
     */
    static bool insert_clause(std::unique_ptr<Clause> &&new_clause, UniqueUnorderedSet<Clause> &destination);

    /**
     * @brief Perform binary resolution on a pair of Clauses.
     * @details Over the Cartesian product of Literals in the given Clauses, we search for and produce
     * Resolvents as follows: <ol> <li>Attempt to unify the LHS Literal with the negation of the RHS
     * Literal;</li> <li>If a (most general) unifier is found, collect the set of Literals (excluding the two
     * source objects) under the application of the MGU;</li> <li>Attempt to derive a tautology through
     * factoring; see @ref factor_literals;</li> <li>If a tautology cannot be derived, construct a Resolvent
     * containing the clause as its resolution, also keeping track of the MGU and source Clauses.</li>
     *  </ol>
     * @param lhs_node The source LHS node containing Literals to subject to binary resolution.
     * @param rhs_node The source RHS node containing Literals to subject to binary resolution.
     * @return The generated Resolvents, and owning containers of resolutions referenced by the Resolvents.
     */
    [[nodiscard]] std::vector<std::pair<Resolvent, std::unique_ptr<Clause>>> find_resolvents(
            const ProofTreeNode *lhs_node, const ProofTreeNode *rhs_node);

    static const log4cxx::LoggerPtr prover_logger;
    static const log4cxx::LoggerPtr resolution_logger;
    static const log4cxx::LoggerPtr factoring_logger;

    std::shared_ptr<SymbolRepository> symbol_repository;
```

```
    FVIKnowledgeBase base_clauses;

    BidirectionalUnificationVisitor unifier;
    BidirectionalUnificationVisitor factoring_unifier;

    UnificationApplicationVisitor applicator;
    UnificationApplicationVisitor factoring_applicator;
};

} // namespace optifol

#endif // OPTIFOL_PROVER_HPP
```

1.9.9 QueryResult.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the FOL Resolution Knowledge Base
 * @author Oliver Dixon
 * @date 2025-09-14
 * @version Development
 */

#ifndef OPTIFOL_QUERYRESULT_HPP
#define OPTIFOL_QUERYRESULT_HPP

#include <deque>

#include "FVIKnowledgeBase.hpp"
#include "Resolvent.hpp"

namespace optifol
{

/**
 * @class QueryResult
 * @brief A QueryResult is a deduction environment for a single query on a FOL KB. It contains an overall
 * result, any derived clauses introduced during deduction, and some metadata.
 */
struct QueryResult
{
    enum class ConjectureStatus
    {
        NotExecuted,
        TimedOut,
        Consistent,
        Inconsistent
    };

    explicit QueryResult(std::shared_ptr<SymbolRepository> symbol_repository) :
        introduced_clauses(std::move(symbol_repository))
    {
    }

    ConjectureStatus outcome = ConjectureStatus::NotExecuted;
    std::size_t elapsed_step_count = 1;

    FVIKnowledgeBase introduced_clauses;
    std::deque<Resolvent> relations;
    const Resolvent *terminating_resolvent = nullptr;
};

} // namespace optifol

#endif // OPTIFOL_QUERYRESULT_HPP
```

1.9.10 Resolvent.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */
```

```cpp
/**
 * @file
 * @brief Class implementation for the FOL Resolution resolvent
 * @author Oliver Dixon
 * @date 2026-01-25
 * @version Development
 */

#include "Resolvent.hpp"

#include <algorithm>
#include <ranges>
#include <utility>

namespace optifol
{

Resolvent::Resolvent(const ProofTreeNode *const lhs_parent, const ProofTreeNode *const rhs_parent,
        Unifier unifier, const Clause *resolution) :
    ProofTreeNode(lhs_parent, rhs_parent),
    unifier(std::move(unifier)),
    resolution(resolution)
{
}

std::size_t Resolvent::hash() const noexcept
{
    return resolution->hash();
}

std::ostream &Resolvent::serialise(std::ostream &ostream) const
{
    return ostream << *resolution;
}

bool Resolvent::operator==(const Resolvent &other) const
{
    return resolution == other.resolution;
}

bool Resolvent::is_unit() const noexcept
{
    return resolution->is_unit();
}

const Clause *Resolvent::observe_node() const noexcept
{
    return resolution;
}

std::optional<const Unifier *> Resolvent::observe_edge() const noexcept
{
    return &unifier;
}

bool Resolvent::operator<(const Resolvent &other) const noexcept
{
    const auto we_have_unit = observe_lhs_parent()->is_unit() || observe_rhs_parent()->is_unit();
    const auto other_has_unit =
            other.observe_lhs_parent()->is_unit() || other.observe_rhs_parent()->is_unit();

    if (we_have_unit && !other_has_unit)
        return false; // LHS has higher priority.

    if (!we_have_unit && other_has_unit)
        return true; // RHS has higher priority.

    // Equal priority. For resolvents, we don't care about the exact ordering provided by Clause::operator<.
    return false;
}

} // namespace optifol
```

1.9.11 Resolvent.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
```

```
 */

/**
 * @file
 * @brief Class specification for the FOL Resolution resolvent
 * @author Oliver Dixon
 * @date 2026-01-25
 * @version Development
 */

#ifndef OPTIFOL_RESOLVENT_HPP
#define OPTIFOL_RESOLVENT_HPP

#include "ProofTreeNode.hpp"
#include "ResolventQueue.hpp"
#include "Unifier.hpp"

namespace optifol
{

/**
 * @class Resolvent
 * @brief A Resolvent is a ProofTreeNode containing a LHS and RHS parent, a resolvent Clause deduced from the
 * parents, and a Unifier map to induce the resolution.
 */
class Resolvent : public IHashable,
                  public ProofTreeNode
{
public:
    Resolvent(const ProofTreeNode *lhs_parent, const ProofTreeNode *rhs_parent, Unifier unifier,
            const Clause *resolution);

    Resolvent(Resolvent &&) = default;
    Resolvent &operator=(Resolvent &&) = default;

    [[nodiscard]] bool operator<(const Resolvent &other) const noexcept;

    [[nodiscard]] std::size_t hash() const noexcept override;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] bool operator==(const Resolvent &other) const;

    [[nodiscard]] bool is_unit() const noexcept override;

    [[nodiscard]] const Clause *observe_node() const noexcept override;

    [[nodiscard]] std::optional<const Unifier *> observe_edge() const noexcept override;

private:
    friend void ResolventQueue::push(Resolvent, std::unique_ptr<Clause> &&);

    Unifier unifier;
    const Clause *resolution;
};

} // namespace optifol

#endif // OPTIFOL_RESOLVENT_HPP
```

1.9.12 ResolventQueue.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the priority queue of Resolvents
 * @author Oliver Dixon
 * @date 2026-01-29
 * @version Development
 */

#include "ResolventQueue.hpp"

#include <algorithm>
#include <cassert>
```

```cpp
#include "FVIKnowledgeBase.hpp"
#include "Resolvent.hpp"

namespace optifol
{

void ResolventQueue::push(Resolvent element, std::unique_ptr<Clause> &&resolution)
{
    assert(element.observe_node() == resolution.get());

    auto [resolution_it, was_new] = resolutions.insert(std::move(resolution));
    if (!was_new) {
        /*
         * If the insertion could not occur, one of two things has happened:
         *
         *  1. The insertion genuinely failed, indicating a runtime error; or
         *
         *  2. The resolution clause of the incoming resolvent is already owned by us. Therefore, we update
         * the pointer (using the "friend" access rights conferred by Resolvent) such that it observes a
         * consistently owned clause. Once this process is repeated for all resolvents, there will be a
         * de-duplicated set of resolution clauses (either owned by us or transferred into a QueryResult) to
         * which all resolvents correctly refer.
         */
        if (resolution_it == resolutions.end())
            throw std::runtime_error(
                    "The ownership of the incoming resolution clause could not be transferred into "
                    "the queue.");
        element.resolution = resolution_it->get();
    }

    elements.push_back(std::move(element));
    std::ranges::push_heap(elements, comparator); // NOLINT(*-use-transparent-functors)
}

Resolvent ResolventQueue::pop()
{
    if (empty())
        throw std::runtime_error("The ResolventQueue is empty.");

    std::ranges::pop_heap(elements, comparator); // NOLINT(*-use-transparent-functors)
    auto resolvent = std::move(elements.back());
    elements.pop_back();
    return resolvent;
}

std::unique_ptr<Clause> ResolventQueue::extract_resolution(const Resolvent &resolvent)
{
    const auto resolution_it = resolutions.find(*resolvent.observe_node());
    if (resolution_it == resolutions.end())
        return {};

    return std::move(resolutions.extract(resolution_it).value());
}

bool ResolventQueue::store_resolution(std::unique_ptr<Clause> &&resolution)
{
    return resolutions.insert(std::move(resolution)).second;
}

bool ResolventQueue::empty() const noexcept
{
    return elements.empty();
}

std::size_t ResolventQueue::size() const noexcept
{
    return elements.size();
}

void ResolventQueue::dump(std::deque<Resolvent> &resolvent_destination, FVIKnowledgeBase &clause_destination)
{
    while (!empty()) {
        auto resolvent = pop();
        auto resolution = extract_resolution(resolvent);
        resolvent_destination.push_back(std::move(resolvent));

        if (resolution != nullptr)
            clause_destination.add_clause(std::move(resolution));
    }
}
```

```
} // namespace optifol
```

1.9.13 ResolventQueue.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the priority queue of Resolvents
 * @author Oliver Dixon
 * @date 2026-01-29
 * @version Development
 */

#ifndef OPTIFOL_RESOLVENTQUEUE
#define OPTIFOL_RESOLVENTQUEUE

#include <deque>
#include <memory>

#include "../IR/Sentences/Clause.hpp"

namespace optifol
{
class FVIKnowledgeBase;
class Resolvent;

/**
 * @class ResolventQueue
 * @brief The ResolventQueue is a specialised @ref std::priority_queue style of max-heap enabling ordered
 * retrieval of Resolvents and temporary unstructured ownership of their corresponding resolution clauses.
 */
class ResolventQueue
{
public:
    /**
     * @brief Push a new Resolvent to the queue.
     * @param element The Resolvent value to push.
     * @param resolution An owning container for the resolution Clause i.a.w. the given Resolvent.
     * @pre The resolution owned by the given Resolvent must refer to the resolution Clause in the given
     *  @ref std::unique_ptr.
     */
    void push(Resolvent element, std::unique_ptr<Clause> &&resolution);

    /**
     * @brief Pop and return the maximal Resolvent from the queue, where order is determined by the @ref
     * std::less functor on Resolvent. This does not change ownership of resolution Clauses.
     * @return The maximal Resolvent owned by the queue.
     * @throws std::runtime_error if the resolvent queue is empty.
     */
    Resolvent pop();

    /**
     * @brief Extract and transfer ownership of the resolution Clause referred to by the given Resolvent.
     * @param resolvent The Resolvent which resolves to the desired Clause.
     * @return An owning container for the resolution Clause.
     */
    std::unique_ptr<Clause> extract_resolution(const Resolvent &resolvent);

    /**
     * @brief Confer ownership of a resolution Clause into the ResolventQueue.
     * @param resolution The resolution to store.
     */
    bool store_resolution(std::unique_ptr<Clause> &&resolution);

    /**
     * @brief Checks emptiness state of the queue
     * @return Is the queue empty?
     */
    [[nodiscard]] bool empty() const noexcept;

    /**
     * @brief Checks size of the queue
     * @return How many Resolvents are on the queue?
     */
```

```
    [[nodiscard]] std::size_t size() const noexcept;

    void dump(std::deque<Resolvent> &resolvent_destination, FVIKnowledgeBase &clause_destination);

private:
    std::deque<Resolvent> elements;
    std::less<Resolvent> comparator{};
    UniqueUnorderedSet<Clause> resolutions;
};

} // namespace optifol

#endif // OPTIFOL_RESOLVENTQUEUE
```

1.9.14 Unifier.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Unifier substitution map
 * @author Oliver Dixon
 * @date 2026-01-26
 * @version Development
 */

#include "Unifier.hpp"

#include <ranges>

namespace optifol
{

Unifier::Unifier() = default;

Unifier::Unifier(RawUnorderedMap<const Variable, const IProcessedTerm *> unifier) :
    unifier(std::move(unifier))
{
}

bool Unifier::operator==(const Unifier &other) const
{
    return unifier == other.unifier;
}

std::ostream &Unifier::serialise(std::ostream &ostream) const
{
    ostream << "{ ";

    const auto unifier_count_bound = unifier.size() - 1;

    for (const auto &&[idx, mapping]: std::ranges::enumerate_view(unifier)) {
        const auto [var, sub] = mapping;
        ostream << *var << '/' << *sub;
        if (static_cast<decltype(unifier_count_bound)>(idx) < unifier_count_bound)
            ostream << ", ";
    }

    return ostream << " }";
}

std::vector<std::string> Unifier::split_serialise() const
{
    std::vector<std::string> substitution_strings;
    substitution_strings.reserve(unifier.size());

    for (const auto &mapping: unifier) {
        const auto [var, sub] = mapping;
        std::ostringstream ostream;
        ostream << *var << " / " << *sub;
        substitution_strings.push_back(std::move(ostream.str()));
    }

    return substitution_strings;
}
```

```
} // namespace optifol
```

1.9.15 Unifier.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Unifier substitution map
 * @author Oliver Dixon
 * @date 2026-01-23
 * @version Development
 */

#ifndef OPTIFOL_UNIFIER_HPP
#define OPTIFOL_UNIFIER_HPP

#include "../IR/Terms/Variable.hpp"
#include "../Optifol.hpp"

namespace optifol
{

class IProcessedTerm;

/**
 * @class Unifier
 * @brief A thin wrapper for Variable-to-term substitutions; typically generated by
 * BidirectionalUnificationVisitor and consumed by UnificationApplicationVisitor.
 */
struct Unifier : ISerialisable
{
    /**
     * @brief Construct an empty Unifier with no substitutions
     */
    Unifier();

    /**
     * @brief Construct a Unifier with an initial set of substitutions.
     * @param unifier The initial substitution set, mapping unique Variable term to IProcessedTerm objects.
     */
    // ReSharper disable once CppNonExplicitConvertingConstructor
    Unifier(RawUnorderedMap<const Variable, const IProcessedTerm *> unifier);

    /**
     * @brief Compare two Unifier instances according to the unordered substitutions.
     * @param other The other Unifier.
     * @return Do the two Unifier instances represent the same set of substitutions?
     */
    [[nodiscard]] bool operator==(const Unifier &other) const;

    /**
     * @brief Serialise the @f$ N @f$ Unifier substitutions @f$ \left( v_i \mapsto t_i \right) @f$ in the form
     *  @f$ \left\{ v_1 / t_1, \ldots, v_N / t_N \right\} @f$ to the given output stream.
     * @param ostream The destination output stream.
     * @return The populated output stream.
     */
    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] std::vector<std::string> split_serialise() const;

    /**
     * @brief The Unifier contents, wherein each entry represents a mapping between a Variable and an
     * IProcessedTerm.
     * @note The key and value are observing pointers to a SymbolRepository, and as such, require all the
     * usual lifetime guarantees conferred by a non-ephemeral SymbolRepository.
     */
    RawUnorderedMap<const Variable, const IProcessedTerm *> unifier;
};

} // namespace optifol

#endif // OPTIFOL_UNIFIER_HPP
```

1.10 Interpreter

1.10.1 FOLLexer.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

#ifndef FOLLEXER_HPP
#define FOLLEXER_HPP

#pragma clang diagnostic push
#pragma ide diagnostic ignored "OCUnusedStructInspection"
#pragma ide diagnostic ignored "NotImplementedFunctions"
#pragma ide diagnostic ignored "OCUnusedGlobalDeclarationInspection"

#include "FOLParser.hpp"

#ifndef __FLEX_LEXER_H
#undef yyFlexLexer
#define yyFlexLexer FOLFlexLexer
#include <FlexLexer.h>
#endif

namespace optifol
{

class FOLLexer : public yyFlexLexer
{
public:
    FOLLexer(std::istream &yy_in, std::ostream &yy_out) :
        yyFlexLexer(yy_in, yy_out)
    {
    }

    int lex(FOLParser::semantic_type *yylval);
};

} // namespace optifol

#pragma clang diagnostic pop

#endif
```

1.10.2 FOLLexer.l

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

%{
#include "FOLParser.hpp"
#include "FOLLexer.hpp"

#undef YY_DECL
#define YY_DECL int optifol::FOLLexer::lex(optifol::FOLParser::semantic_type *yylval)
%}

%option c++ interactive noyywrap noyylineno
%option yyclass="FOLLexer"
%option prefix="FOL"

lowercase_sequence ([a-z]+)
uppercase_sequence ([A-Z]+)
whitespace         ([ \t]+)
symbol_prefix      "%"
function_prefix    "#"
character_sequence ([a-zA-Z]+)

%%

{function_prefix}{character_sequence}   yylval->emplace<std::string>(YYText()); return
    FOLParser::token::Function;
{lowercase_sequence}                    yylval->emplace<std::string>(YYText()); return
    FOLParser::token::Variable;
{uppercase_sequence}                    yylval->emplace<std::string>(YYText()); return
```

```
    FOLParser::token::Predicate;

{symbol_prefix}"U"    return FOLParser::token::Universal;
{symbol_prefix}"E"    return FOLParser::token::Existential;

"&"      return FOLParser::token::Conjunction;
"|"      return FOLParser::token::Disjunction;
"~"      return FOLParser::token::Negation;
"=>"     return FOLParser::token::Implication;
"<=>"    return FOLParser::token::Biconditional;

"(" return FOLParser::token::LeftParenthesis;
")" return FOLParser::token::RightParenthesis;
"," return FOLParser::token::Comma;

\n           return FOLParser::token::End;
<<EOF>>      return FOLParser::token::End;
{whitespace} /* Ignore whitespace */

%%
```

1.10.3 FOLParser.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification of the exposed first-order logic parser
 * @date 2024-11-21
 * @author Oliver Dixon <od641@york.ac.uk>
 * @note This file depends on build-time generated source from Flex and Bison.
 */

#ifndef FOLPARSER_HPP
#define FOLPARSER_HPP

#include <memory>

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Weverything"
#include <generated/BaseFOLParser.hpp> // Generated by Bison
#pragma clang diagnostic pop

#include "../Exceptions/ParseError.hpp"
#include "../IR/MutableVariants/Sentences/IMutableSentence.hpp"

namespace optifol
{

/**
 * @brief The first-order logic parser encapsulates the raw Bison-generated parser and provides various safety
 * and convenience mechanisms.
 */
class FOLParser : public impl::BaseFOLParser
{
public:
    /**
     * @brief Instantiate a new first-order logic parser, to be supplied by the given lexer
     * @param lexer The Flex-created lexer with which the parser should be acquainted
     */
    explicit FOLParser(FOLLexer *lexer) :
        BaseFOLParser(lexer)
    {
    }

    void error(const std::string &msg) override
    {
        throw ParseError(msg, 0);
    }

    /**
     * @brief Reports a new fully parsed root sentence, typically from another parser-like source
     * @param sentence_root The parsed FOL sentence
     */
    [[maybe_unused]] void register_sentence(std::unique_ptr<MutableSentenceRoot> &&sentence_root)
    {
```

```
        last_parsed_root = std::move(sentence_root);
    }

    [[nodiscard]] std::unique_ptr<MutableSentenceRoot> retrieve_sentence()
    {
        auto sentence = std::move(last_parsed_root);
        last_parsed_root = nullptr;
        return sentence;
    }

private:
    std::unique_ptr<MutableSentenceRoot> last_parsed_root;
};

} // namespace optifol

#endif
```

1.10.4 FOLParser.y

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

%require "3.7.4"
%language "C++"

%define api.parser.class {BaseFOLParser}
%define api.namespace {optifol::impl}
%define api.prefix {FOL}
%define api.value.type variant
%define parse.error detailed
%defines
%skeleton "lalr1.cc"
%parse-param {FOLLexer* scanner}

%code requires
{
    #include "../IR/MutableVariants/Terms/MutableVariable.hpp"
    #include "../IR/MutableVariants/Terms/MutableFunction.hpp"

    #include "../IR/MutableVariants/Sentences/MutablePredicate.hpp"
    #include "../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
    #include "../IR/MutableVariants/Sentences/MutableQuantified.hpp"
    #include "../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"

    namespace optifol
    {
        class FOLLexer;
    }
}

%code
{
    #include "FOLLexer.hpp"
    #define yylex(x) scanner->lex(x)
}

%token <std::string> Function
%token <std::string> Variable
%token <std::string> Predicate

%token Universal Existential
%token Conjunction Disjunction Negation Implication Biconditional
%token LeftParenthesis RightParenthesis Comma
%token End

%left Biconditional
%left Implication
%left Disjunction
%left Conjunction
%right Negation

%type <IMutableSentence *> sentence
%type <IMutableTerm *> term
%type <std::vector<std::unique_ptr<IMutableTerm>>> term_vector

%start line
```

```
/*
 * Note the overall strategy for handling internal state during the parse. Raw C pointers are used to refer to
    nodes
 * during the build process, and are only transformed into unique pointers (via std::unique_ptr<T>(T*) or
    T::build(...))
 * at their relative termini. Thus the built model has full ownership semantics, and we don't have to worry
    about Bison
 * generating implicit calls to the default copy constructor, which is deleted for std::unique_ptr.
 */

%%

line :
    sentence End
    {
        static_cast<FOLParser *>(this)->register_sentence(
            MutableSentenceRoot::build(std::unique_ptr<IMutableSentence>($1))
        );

        return 0;
    }
    |
    error
    {
        return -1;
    }
    ;

sentence :
    Universal Variable LeftParenthesis sentence RightParenthesis
    {
        $$ = new MutableQuantified(
            QuantifierTypes::Universal,
            MutableVariable::build($2),
            std::unique_ptr<IMutableSentence>($4)
        );
    }
    |
    Existential Variable LeftParenthesis sentence RightParenthesis
    {
        $$ = new MutableQuantified(
            QuantifierTypes::Existential,
            MutableVariable::build($2),
            std::unique_ptr<IMutableSentence>($4)
        );
    }
    |
    Predicate LeftParenthesis term_vector RightParenthesis
    {
        $$ = new MutablePredicate($1, std::move($3));
    }
    |
    Negation sentence
    {
        $2->flip_polarity();
        $$ = $2;
    }
    |
    sentence Conjunction sentence
    {
        $$ = new MutableBinaryConnected(
            BinaryOperatorTypes::Conjunction,
            std::unique_ptr<IMutableSentence>($1),
            std::unique_ptr<IMutableSentence>($3)
        );
    }
    |
    sentence Disjunction sentence
    {
        $$ = new MutableBinaryConnected(
            BinaryOperatorTypes::Disjunction,
            std::unique_ptr<IMutableSentence>($1),
            std::unique_ptr<IMutableSentence>($3)
        );
    }
    |
    sentence Implication sentence
    {
        $$ = new MutableBinaryConnected(
            BinaryOperatorTypes::Implication,
```

```
            std::unique_ptr<IMutableSentence>($1),
            std::unique_ptr<IMutableSentence>($3)
        );
    }
    |
    sentence Biconditional sentence
    {
        $$ = new MutableBinaryConnected(
            BinaryOperatorTypes::Biconditional,
            std::unique_ptr<IMutableSentence>($1),
            std::unique_ptr<IMutableSentence>($3)
            );
        }
    |
    LeftParenthesis sentence RightParenthesis
    {
        $$ = $2;
    }
    ;

term_vector :
    %empty
    {
        $$ = std::vector<std::unique_ptr<IMutableTerm>>();
    }
    |
    term
    {
        $$ = std::vector<std::unique_ptr<IMutableTerm>>();
        $$.push_back(std::unique_ptr<IMutableTerm>($1));
    }
    |
    term_vector Comma term
    {
        $$ = std::move($1);
        $$.push_back(std::unique_ptr<IMutableTerm>($3));
    }
    ;

term :
    Function
    {
        $$ = new MutableFunction($1);
    }
    |
    Function LeftParenthesis term_vector RightParenthesis
    {
        $$ = new MutableFunction($1, std::move($3));
    }
    |
    Variable
    {
        $$ = new MutableVariable($1);
    }
    ;

%%

void optifol::impl::BaseFOLParser::error(const std::string& msg)
{
    static_cast<FOLParser *>(this)->error(msg);
}
```

1.11 IR

1.11.1 SymbolRepository.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the SymbolRepository class
 * @author Oliver Dixon
 * @date 2025-06-13
 * @version Development
 */
```

```cpp
#include "SymbolRepository.hpp"

namespace optifol
{

const IProcessedTerm *SymbolRepository::get_symbol_handle(const IProcessedTerm &term) const
{
    const auto it = terms.find(term);
    if (it == terms.cend()) {
        const auto variable_try = dynamic_cast<const Variable *>(&term);
        return variable_try == nullptr ? nullptr : get_symbol_handle(*variable_try);
    }

    return it->get();
}

const Variable *SymbolRepository::get_symbol_handle(const Variable &variable) const
{
    const auto it = variables.find(variable);
    if (it == variables.cend())
        return nullptr;

    return it->get();
}

bool SymbolRepository::operator==(const SymbolRepository &other) const noexcept
{
    return sentences == other.sentences && terms == other.terms;
}

void SymbolRepository::inherit_repository(std::unique_ptr<SymbolRepository> &&other)
{
    inherit_set(sentences, other->sentences);
    inherit_set(terms, other->terms);
    inherit_set(variables, other->variables);
}

void SymbolRepository::increment_sentence_count() noexcept
{
    ++sentence_count;
}

unsigned int SymbolRepository::get_sentence_count() const noexcept
{
    return sentence_count;
}

} // namespace optifol
```

1.11.2 SymbolRepository.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the SymbolRepository class
 * @author Oliver Dixon
 * @date 2025-06-13
 * @version Development
 */

#ifndef SYMBOLREPOSITORY_HPP
#define SYMBOLREPOSITORY_HPP

#include <memory>
#include <unordered_set>

#include "../Optifol.hpp"
#include "Sentences/IProcessedSentence.hpp"
#include "Terms/IProcessedTerm.hpp"
#include "Terms/Variable.hpp"

namespace optifol
{
```

```
/**
 * @class SymbolRepository
 * @brief A SymbolRepository persists a single set of sentenceNodes and ITerms with guaranteed lifetimes. The
 *  repository can be populated and queried, exposing constant pointers, but elements cannot be removed or
 * destructed until the entire repository is destructed.
 */
class SymbolRepository
{
public:
    /**
     * @brief Add a generic term to the repository
     * @tparam TermType The type of term pointer to return
     * @param term A \ref std::unique_ptr wrapper of the ITerm whose ownership should be transferred.
     * @return A constant handle to the term in the repository
     * @throws std::runtime_error An equivalent term does not exist in the repository, and could not be added.
     * @note If the supplied term is hash-equal to an existing term held by the repository, the repository is
     * unchanged.
     */
    template<typename TermType = IProcessedTerm>
        requires std::derived_from<TermType, IProcessedTerm>
    const TermType *add_symbol(std::unique_ptr<TermType> &&term)
    {
        const auto find_it = terms.find(*term);
        if (find_it != terms.cend()) {
            // If a hash-equal element is already in the set, attempt to downcast to the requested type and
            // return.
            const auto downcast_ptr = dynamic_cast<const TermType *>(find_it->get());
            if (downcast_ptr == nullptr)
                throw std::runtime_error("Cannot add term " + std::string(term->get_disambiguated_name()) +
                        ": a matching term of a different type already exists in the repository.");
            return downcast_ptr;
        }

        // Otherwise, add the new element by transferring ownership to the set.
        const auto [inserted_it, success] = terms.insert(std::move(term));
        if (!success)
            throw std::runtime_error("Cannot add term: insertion failed.");

        /*
         * This is a bit dodgy, as the compiler isn't enforcing semantics correctness of the pointer cast, as
         * would be suggested by use of static_cast. It could equally be cast to any other pointer and be
         * undefined at dereference. But the STL standard provides this guarantee, as if the insertion is
         * successful, the returned iterator is defined to detain a type of the key, i.e.
         * std::unique_ptr<IProcessedTerm>. We know from the function signature that our term holds a pointer
         * of type TermType.
         */
        return static_cast<const TermType *>(inserted_it->get());
    }

    /**
     * @brief Add a sentence to the repository
     * @tparam SentenceType The type of sentence pointer to return
     * @param sentence A \ref std::unique_ptr wrapper of the sentence whose ownership should be transferred.
     * @return A constant handle to the sentence in the repository
     * @throws std::runtime_error An equivalent sentence does not exist in the repository, and could not be
     * added.
     * @note If the supplied sentence is hash-equal to an existing sentence held by the repository, the
     * repository is unchanged.
     */
    template<typename SentenceType = IProcessedSentence>
        requires std::derived_from<SentenceType, IProcessedSentence>
    const SentenceType *add_symbol(std::unique_ptr<SentenceType> &&sentence)
    {
        const auto find_it = sentences.find(*sentence);
        if (find_it != sentences.cend()) {
            const auto downcast_ptr = dynamic_cast<const SentenceType *>(find_it->get());
            if (downcast_ptr == nullptr)
                throw std::runtime_error(
                        "Cannot add sentence: a matching sentence of a different type already exists "
                        "in the repository.");
            return downcast_ptr;
        }

        const auto [inserted_it, success] = sentences.insert(std::move(sentence));
        if (!success)
            throw std::runtime_error("Cannot add sentence: insertion failed.");

        return static_cast<const SentenceType *>(inserted_it->get());
    }

    /**
```

```
     * @brief Retrieves a handle to an immutable term symbol owned by the repository
     * @param term A hash-equal ITerm to the target term
     * @return A constant handle to the term, if a suitable match exists in the repository. Otherwise, an
     * empty @ref std::optional.
     */
    const IProcessedTerm *get_symbol_handle(const IProcessedTerm &term) const;

    /**
     * @brief Retrieves a handle to an immutable sentence symbol owned by the repository
     * @tparam SentenceType The type of sentence pointer to return
     * @param sentence A hash-equal sentence to the target sentence
     * @return A constant handle to the sentence, if a suitable match exists in the repository. Otherwise, an
     * empty
     *  @ref std::optional.
     */
    template<typename SentenceType = IProcessedSentence>
        requires std::derived_from<SentenceType, IProcessedSentence>
    const SentenceType *get_symbol_handle(const IProcessedSentence &sentence) const
    {
        const auto it = sentences.find(sentence);
        const SentenceType *downcast_ptr = nullptr;

        if (it == sentences.cend() ||
                (downcast_ptr = dynamic_cast<const SentenceType *>(it->get())) == nullptr)
            return nullptr;

        return downcast_ptr;
    }

    const Variable *get_symbol_handle(const Variable &variable) const;

    /**
     * @brief Determine equality between two SymbolRepository objects according to their stored sentences and
     * terms.
     * @param other The SymbolRepository with which to compare elements.
     * @return Do the SymbolRepository objects store the same items?
     */
    [[nodiscard]] bool operator==(const SymbolRepository &other) const noexcept;

    /**
     * @brief Move all symbols (sentences, non-variable and variable terms) from the given SymbolRepository
     * into this one.
     * @param other The target repository, the entirety of which is to be consumed by the inheritor.
     */
    void inherit_repository(std::unique_ptr<SymbolRepository> &&other);

    void increment_sentence_count() noexcept;

    [[nodiscard]] unsigned int get_sentence_count() const noexcept;

private:
    /**
     * @brief Steal @ref std::unique_ptr objects from the given UniqueUnorderedSet source container and move
     * into the given destination.
     * @tparam Element The type of elements owned by the members of the set.
     * @param dest The destination set.
     * @param src The source set.
     */
    template<class Element>
    static void inherit_set(UniqueUnorderedSet<Element> &dest, UniqueUnorderedSet<Element> &src)
    {
        for (auto node_handle_it = src.begin(); node_handle_it != src.end();) {
            auto borrowed_node = src.extract(node_handle_it++);
            dest.insert(std::move(borrowed_node));
        }
    }

    UniqueUnorderedSet<IProcessedSentence> sentences;
    UniqueUnorderedSet<IProcessedTerm> terms;
    UniqueUnorderedSet<Variable> variables;

    unsigned int sentence_count = 0;
};

template<>
inline const Variable *SymbolRepository::add_symbol(std::unique_ptr<Variable> &&term)
{
    const auto find_it = variables.find(*term);
    if (find_it != variables.cend())
        return find_it->get();
```

```
    const auto [inserted_it, success] = variables.insert(std::move(term));
    if (!success)
        throw std::runtime_error("Cannot add variable: insertion failed.");

    return inserted_it->get();
}

} // namespace optifol

#endif
```

1.11.3 MutableVariants

1.11.3.1 OwningBuildable.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class template specification for ownable-buildable Optifol objects
 * @author Oliver Dixon
 * @date 2025-06-08
 * @version Development
 */

#ifndef OWNINGBUILDABLE_HPP
#define OWNINGBUILDABLE_HPP

#include <memory>

namespace optifol
{

/**
 * @class OwningBuildable
 * @brief The OwningBuildable class provides a CRTP-templated base to classes wishing to be trivially wrapped
 * into a
 *  @ref std::unique_ptr. This is useful to express explicit ownership and transfers thereof.
 * @tparam BuildType The concrete class type to build
 */
template<typename BuildType>
class OwningBuildable
{
public:
    /**
     * @brief Construct a class with the given constructor arguments in a unique pointer
     * @tparam PointerType The stored pointer type of the constructed unique pointer; must be a base of, or
     * equal to, the class BuildType template parameter.
     * @tparam CtorArgs The template parameter pack of argument types to forward to the BuildType constructor.
     * @param args The argument values, corresponding to the given types, to perfectly forward to the
     * constructor
     * @return An instance of the BuildType class, constructed according to the given arguments, wrapped in a
     * unique pointer containing a pointer to the specified type.
     */
    template<typename PointerType = BuildType, typename... CtorArgs>
        requires std::derived_from<BuildType, PointerType>
    static std::unique_ptr<PointerType> build(CtorArgs &&...args)
    {
        return std::make_unique<BuildType>(std::forward<CtorArgs>(args)...);
    }
};

} // namespace optifol

#endif
```

1.11.3.2 Sentences

1.11.3.2.1 IMutableSentence.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */
```

```cpp
/**
 * @file
 * @brief Class specification for the generic mutable IR Sentence interface
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#ifndef IMUTABLESENTENCE_HPP
#define IMUTABLESENTENCE_HPP

#include <memory>

#include "../../Sentences/ISentence.hpp"

namespace optifol
{

class MutatingSentenceVisitorBase;
class IObservingNodeVisitor;
class RepositoryBuildingVisitor;
class IProcessedSentence;

/**
 * @class IMutableSentence
 * @brief A mutable sentence is a sentence IR node with properties useful during parsing and normalisation,
 * prior to formal analysis.
 * @details A mutable sentence, in general, has the following properties:
 *  <ul>
 *      <li>
 *          Any child nodes (e.g. arguments to a MutablePredicate, bound sentences to MutableQuantified, etc.)
 * are strictly owned by explicitly transferable containers, typically a @ref std::unique_ptr. Immutable
 * observing pointers can be extracted without requiring a non-constant object, but for mutation ownership
 * needs to be explicitly stolen from the sentence node and later returned through <code>steal</code> and
 * <code>put</code> member functions.
 *      </li>
 *      <li>
 *          POD metadata (e.g. display names and operator types for MutableBinaryConnected) are changeable on
 *          non-constant objects, and need not persist the same values for the lifetime of the object.
 *      </li>
 *  </ul>
 */
class IMutableSentence : public ISentence
{
public:
    /**
     * @brief Perform a deep-copy of the sentence and produce a copy wrapped in a transferable @ref
     * std::unique_ptr.
     * @return The container containing the copied sentence.
     */
    [[nodiscard]] virtual std::unique_ptr<IMutableSentence> clone() const = 0;

    /**
     * @brief Flip the polarity of the instantiated sentence
     * @see ISentence::is_negative_polarity for dual getter
     */
    virtual void flip_polarity() noexcept = 0;

    /**
     * @brief Accept a visitation from a mutating sentence visitor on the object
     * @param visitor The mutating visitor to invoke
     */
    virtual void accept(MutatingSentenceVisitorBase &visitor) = 0;

    /**
     * @brief Accept a visitation from an observing (non-mutating) sentence visitor on the object
     * @param visitor The observing visitor to invoke
     */
    virtual void accept(IObservingNodeVisitor &visitor) const = 0;

    /**
     * @brief Accept a visitation from a mutating RepositoryBuildingVisitor on the object
     * @param visitor The repository-building visitor to invoke
     * @return An observing reference to the ISentence node added to a central SymbolRepository object
     */
    virtual const IProcessedSentence *accept(RepositoryBuildingVisitor &visitor) = 0;

    /**
     * @brief Test equality between two IMutableSentence instances.
     * @param other The other IMutableSentence instance with which to test for equality.
```

```
     * @return Are the two sentences of the same concrete type and equal?
     */
    [[nodiscard]] virtual bool operator==(const IMutableSentence &other) const noexcept = 0;
};

} // namespace optifol

#endif
```

1.11.3.2.2 MutableBinaryConnected.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the mutable binary-Connected Sentence IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#include "MutableBinaryConnected.hpp"

#include "../../../Visitors/MutableTargets/Observers/IObservingNodeVisitor.hpp"
#include "../../../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../../../Visitors/MutableTargets/Sentences/MutatingSentenceVisitorBase.hpp"

namespace optifol
{

MutableBinaryConnected::MutableBinaryConnected(const BinaryOperatorTypes operator_type,
        std::unique_ptr<IMutableSentence> &&lhs, std::unique_ptr<IMutableSentence> &&rhs,
        const bool is_positive) :
    operator_type(operator_type),
    lhs(std::move(lhs)),
    rhs(std::move(rhs)),
    is_positive(is_positive)
{
}

std::unique_ptr<IMutableSentence> MutableBinaryConnected::clone() const
{
    return std::make_unique<MutableBinaryConnected>(operator_type, lhs->clone(), rhs->clone(), is_positive);
}

void MutableBinaryConnected::flip_polarity() noexcept
{
    is_positive = !is_positive;
}

bool MutableBinaryConnected::is_negative_polarity() const noexcept
{
    return !is_positive;
}

void MutableBinaryConnected::accept(MutatingSentenceVisitorBase &visitor)
{
    visitor.visit(*this);
}

BinaryOperatorTypes MutableBinaryConnected::get_operator_type() const noexcept
{
    return operator_type;
}

void MutableBinaryConnected::set_operator_type(const BinaryOperatorTypes new_type) noexcept
{
    operator_type = new_type;
}

std::unique_ptr<IMutableSentence> MutableBinaryConnected::take_lhs_operand() noexcept
{
    return std::move(lhs);
}

std::unique_ptr<IMutableSentence> MutableBinaryConnected::take_rhs_operand() noexcept
```

```
{
    return std::move(rhs);
}

const IMutableSentence *MutableBinaryConnected::observe_lhs_operand() const noexcept
{
    return lhs.get();
}

const IMutableSentence *MutableBinaryConnected::observe_rhs_operand() const noexcept
{
    return rhs.get();
}

void MutableBinaryConnected::put_lhs_operand(std::unique_ptr<IMutableSentence> &&operand) noexcept
{
    lhs = std::move(operand);
}

void MutableBinaryConnected::put_rhs_operand(std::unique_ptr<IMutableSentence> &&operand) noexcept
{
    rhs = std::move(operand);
}

void MutableBinaryConnected::accept(IObservingNodeVisitor &visitor) const
{
    visitor.visit(*this);
}

const IProcessedSentence *MutableBinaryConnected::accept(RepositoryBuildingVisitor &visitor)
{
    return visitor.visit(*this);
}

std::size_t MutableBinaryConnected::hash() const noexcept
{
    return BinaryConnected::hash_binary_connected(
            operator_type, lhs.get(), rhs.get(), is_negative_polarity());
}

std::ostream &MutableBinaryConnected::serialise(std::ostream &ostream) const
{
    return BinaryConnected::serialise_binary_connected(
            ostream, operator_type, lhs.get(), rhs.get(), is_negative_polarity());
}

bool MutableBinaryConnected::operator==(const IMutableSentence &other) const noexcept
{
    const auto other_connected = dynamic_cast<const MutableBinaryConnected *>(&other);
    if (other_connected == nullptr)
        // Other IMutableSentence isn't a MutableBinaryConnected.
        return false;

    return commutative_ptr_compare(
            lhs.get(), rhs.get(), other_connected->lhs.get(), other_connected->rhs.get());
}

} // namespace optifol
```

1.11.3.2.3 MutableBinaryConnected.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the mutable binary-Connected Sentence IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#ifndef MUTABLEBINARYCONNECTED_HPP
#define MUTABLEBINARYCONNECTED_HPP

#include <memory>
```

```cpp
#include "../../Sentences/BinaryConnected.hpp"
#include "../OwningBuildable.hpp"
#include "IMutableSentence.hpp"

namespace optifol
{

/**
 * @class MutableBinaryConnected
 * @brief A MutableBinaryConnected IR node holds and owns two operands semantically joined with an operator.
 *  Metadata and operands are mutable.
 * @see BinaryOperatorTypes for modes of connection
 * @see BinaryConnected for non-owning dual
 */
class MutableBinaryConnected : public IMutableSentence,
                               public OwningBuildable<MutableBinaryConnected>
{
public:
    /**
     * @brief Create a signed mutable binary-connected IR node
     * @param operator_type Operator with which the operands are connected
     * @param lhs Transferred owning container of the left-hand operand
     * @param rhs Transferred owning container of the right-hand operand
     * @param is_positive Should the node be instantiated with in a positive polarity?
     */
    [[maybe_unused]] MutableBinaryConnected(BinaryOperatorTypes operator_type,
            std::unique_ptr<IMutableSentence> &&lhs, std::unique_ptr<IMutableSentence> &&rhs,
            bool is_positive = true);

    [[nodiscard]] std::unique_ptr<IMutableSentence> clone() const override;

    void flip_polarity() noexcept override;

    [[nodiscard]] bool is_negative_polarity() const noexcept override;

    void accept(MutatingSentenceVisitorBase &visitor) override;

    void accept(IObservingNodeVisitor &visitor) const override;

    const IProcessedSentence *accept(RepositoryBuildingVisitor &visitor) override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] bool operator==(const IMutableSentence &other) const noexcept override;

    /**
     * @brief Get the current operator by which the operands are connected
     * @return The binary-connected operator type
     */
    [[nodiscard]] BinaryOperatorTypes get_operator_type() const noexcept;

    /**
     * @brief Replaces the binary operator
     * @param new_type The new binary operator
     */
    void set_operator_type(BinaryOperatorTypes new_type) noexcept;

    /**
     * @brief Steals ownership of the LHS operand from the object to the caller
     * @return The stolen container containing the LHS operand
     * @warning This call transfers ownership outbound
     */
    [[nodiscard]] std::unique_ptr<IMutableSentence> take_lhs_operand() noexcept;

    /**
     * @brief Steals ownership of the RHS operand from the object to the caller
     * @return The stolen container containing the RHS operand
     * @warning This call transfers ownership outbound
     */
    [[nodiscard]] std::unique_ptr<IMutableSentence> take_rhs_operand() noexcept;

    /**
     * @brief Provides an immutable observing pointer to the owned LHS operand
     * @return The LHS operand observer
     * @see @ref std::unique_ptr::get for semantics of observing getter
     */
    [[nodiscard]] const IMutableSentence *observe_lhs_operand() const noexcept;

    /**
```

```
     * @brief Provides an immutable observing pointer to the owned RHS operand
     * @return The LHS operand observer
     * @see @ref std::unique_ptr::get for semantics of observing getter
     */
    [[nodiscard]] const IMutableSentence *observe_rhs_operand() const noexcept;

    /**
     * @brief Transfers ownership of a new LHS operand, overwriting any previously held LHS operand
     * @param operand The new LHS operand owning container
     * @warning This call transfers ownership inbound
     * @see @ref std::unique_ptr::operator= for semantics of swap
     */
    void put_lhs_operand(std::unique_ptr<IMutableSentence> &&operand) noexcept;

    /**
     * @brief Transfers ownership of a new RHS operand, overwriting any previously held LHS operand
     * @param operand The new RHS operand owning container
     * @warning This call transfers ownership inbound
     * @see @ref std::unique_ptr::operator= for semantics of swap
     */
    void put_rhs_operand(std::unique_ptr<IMutableSentence> &&operand) noexcept;

private:
    BinaryOperatorTypes operator_type;
    std::unique_ptr<IMutableSentence> lhs;
    std::unique_ptr<IMutableSentence> rhs;
    bool is_positive;
};

} // namespace optifol

#endif
```

1.11.3.2.4 MutablePredicate.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the mutable Predicate IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#include "MutablePredicate.hpp"

#include "../../../CompositeSerialisationHelpers.hpp"
#include "../../../Visitors/MutableTargets/Observers/IObservingNodeVisitor.hpp"
#include "../../../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../../../Visitors/MutableTargets/Sentences/MutatingSentenceVisitorBase.hpp"
#include "../../Sentences/Literal.hpp"
#include "../Terms/IMutableTerm.hpp"

namespace optifol
{

MutablePredicate::MutablePredicate(
        std::string name, const bool is_positive, std::vector<std::unique_ptr<IMutableTerm>> &&arguments) :
    name(std::move(name)),
    arguments(std::move(arguments)),
    is_positive(is_positive)
{
}

MutablePredicate::MutablePredicate(std::string name, const bool is_positive,
        const std::vector<std::unique_ptr<IMutableTerm>> &arguments) :
    name(std::move(name)),
    is_positive(is_positive)
{
    this->arguments.reserve(arguments.size());
    for (const auto &arg: arguments)
        this->arguments.push_back(arg->clone());
}

MutablePredicate::MutablePredicate(std::string name, std::vector<std::unique_ptr<IMutableTerm>> &&arguments) :
```

```
    MutablePredicate(std::move(name), true, std::move(arguments))
{
}

MutablePredicate::MutablePredicate(
        std::string name, const std::vector<std::unique_ptr<IMutableTerm>> &arguments) :
    MutablePredicate(std::move(name), true, arguments)
{
}

MutablePredicate::MutablePredicate(std::string name) :
    MutablePredicate(std::move(name), true, {})
{
}

std::unique_ptr<IMutableSentence> MutablePredicate::clone() const
{
    std::vector<std::unique_ptr<IMutableTerm>> cloned_arguments;
    cloned_arguments.reserve(arguments.size());
    for (const auto &argument: arguments)
        cloned_arguments.push_back(argument->clone());

    return std::make_unique<MutablePredicate>(name, is_positive, std::move(cloned_arguments));
}

void MutablePredicate::flip_polarity() noexcept
{
    is_positive = !is_positive;
}

bool MutablePredicate::is_negative_polarity() const noexcept
{
    return !is_positive;
}

void MutablePredicate::accept(MutatingSentenceVisitorBase &visitor)
{
    visitor.visit(*this);
}

void MutablePredicate::accept(IObservingNodeVisitor &visitor) const
{
    visitor.visit(*this);
}

const IProcessedSentence *MutablePredicate::accept(RepositoryBuildingVisitor &visitor)
{
    return visitor.visit(*this);
}

std::size_t MutablePredicate::hash() const noexcept
{
    return composite_hash(name, arguments.cbegin(), arguments.cend(), is_negative_polarity());
}

std::ostream &MutablePredicate::serialise(std::ostream &ostream) const
{
    return CompositeSerialisationHelpers::stream_serialise(
            ostream, name, arguments.cbegin(), arguments.cend(), is_negative_polarity());
}

bool MutablePredicate::operator==(const IMutableSentence &other) const noexcept
{
    const auto other_predicate = dynamic_cast<const MutablePredicate *>(&other);
    if (other_predicate == nullptr)
        // Other IMutableSentence isn't a MutablePredicate.
        return false;

    if (name != other_predicate->name)
        // Different superficial names.
        return false;

    const auto argument_count = arguments.size();
    if (argument_count != other_predicate->arguments.size())
        // Different number of arguments.
        return false;

    for (std::size_t argument_idx = 0; argument_idx < argument_count; ++argument_idx) {
        const auto &lhs_arg_ptr = arguments[argument_idx];
        const auto &rhs_arg_ptr = other_predicate->arguments[argument_idx];
```

```
        if (lhs_arg_ptr == nullptr) {
            if (rhs_arg_ptr != nullptr)
                return false;
        } else if (rhs_arg_ptr == nullptr)
            return false;
        else if (*lhs_arg_ptr != *rhs_arg_ptr)
            return false;
    }

    return true;
}

std::string_view MutablePredicate::get_name() const noexcept
{
    return name;
}

const std::vector<std::unique_ptr<IMutableTerm>> &MutablePredicate::observe_arguments() const noexcept
{
    return arguments;
}

std::vector<std::unique_ptr<IMutableTerm>> &MutablePredicate::observe_arguments() noexcept
{
    return arguments;
}

} // namespace optifol
```

1.11.3.2.5 MutablePredicate.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the mutable Predicate IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#ifndef MUTABLEPREDICATE_HPP
#define MUTABLEPREDICATE_HPP

#include <memory>
#include <vector>

#include "../OwningBuildable.hpp"
#include "../Terms/IMutableTerm.hpp"
#include "IMutableSentence.hpp"

namespace optifol
{

class RepositoryBuildingVisitor;

/**
 * @class MutablePredicate
 * @brief A MutablePredicate represents an owning IR node sentence consisting of a display name and zero to
 * many arguments, all of which are owned as transferable @ref std::unique_ptr objects by the node.
 * @see Predicate for the processed, argument-observing dual.
 */
class MutablePredicate : public IMutableSentence,
                         public OwningBuildable<MutablePredicate>
{
public:
    /**
     * @brief Create an owning predicate with an initial set of owned arguments
     * @param name Display name of the predicate
     * @param is_positive Should the predicate be instantiated with positive polarity?
     * @param arguments A moveable ordered container owning the initial arguments
     */
    explicit MutablePredicate(
            std::string name, bool is_positive, std::vector<std::unique_ptr<IMutableTerm>> &&arguments);

    /**
```

```
     * @brief Create an owning predicate with an initial set of owned arguments
     * @param name Display name of the predicate
     * @param is_positive Should the predicate be instantiated with positive polarity?
     * @param arguments A referenced ordered container containing the moveable arguments
     */
    explicit MutablePredicate(std::string name, bool is_positive,
            const std::vector<std::unique_ptr<IMutableTerm>> &arguments = {});

    /**
     * @brief Create an owning unsigned predicate with an initial set of owned arguments
     * @param name Display name of the predicate
     * @param arguments A moveable ordered container owning the initial arguments
     */
    explicit MutablePredicate(std::string name, std::vector<std::unique_ptr<IMutableTerm>> &&arguments);

    /**
     * @brief Create an owning unsigned predicate with an initial set of owned arguments
     * @param name Display name of the predicate
     * @param arguments A referenced ordered container containing the moveable arguments
     */
    explicit MutablePredicate(std::string name, const std::vector<std::unique_ptr<IMutableTerm>> &arguments);

    /**
     * @brief Create an owning unsigned predicate
     * @param name Display name of the predicate
     */
    explicit MutablePredicate(std::string name);

    [[nodiscard]] std::unique_ptr<IMutableSentence> clone() const override;

    void flip_polarity() noexcept override;

    [[nodiscard]] bool is_negative_polarity() const noexcept override;

    void accept(MutatingSentenceVisitorBase &visitor) override;

    void accept(IObservingNodeVisitor &visitor) const override;

    const IProcessedSentence *accept(RepositoryBuildingVisitor &visitor) override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] bool operator==(const IMutableSentence &other) const noexcept override;

    /**
     * @brief Get the display name of the mutable predicate, not including any arguments or metadata
     * @return A view of the predicate symbol name
     */
    [[nodiscard]] std::string_view get_name() const noexcept;

    /**
     * @brief Observe the constant owning ordered argument collection
     * @return The arguments owned by the predicate
     */
    [[nodiscard]] const std::vector<std::unique_ptr<IMutableTerm>> &observe_arguments() const noexcept;

    /**
     * @brief Observe the mutable owning ordered argument collection
     * @return The arguments owned by the predicate
     * @note This non-constant overload is useful for propagation of <code>accept</code> calls on mutating
     * visitors.
     */
    std::vector<std::unique_ptr<IMutableTerm>> &observe_arguments() noexcept;

private:
    const std::string name;
    std::vector<std::unique_ptr<IMutableTerm>> arguments;
    bool is_positive = true;
};

} // namespace optifol

#endif
```

1.11.3.2.6 MutableQuantified.cpp

```
/*
```

```
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the mutable Quantified Sentence IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#include "MutableQuantified.hpp"

#include <utility>

#include "../../../Visitors/MutableTargets/Observers/IObservingNodeVisitor.hpp"
#include "../../../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../../../Visitors/MutableTargets/Sentences/MutatingSentenceVisitorBase.hpp"

namespace optifol
{

MutableQuantified::MutableQuantified(const QuantifierTypes quantifier_type,
        std::unique_ptr<IMutableTerm> &&bound_term, std::unique_ptr<IMutableSentence> &&sentence,
        const bool is_positive) :
    quantifier_type(quantifier_type),
    bound_term(std::move(bound_term)),
    sentence(std::move(sentence)),
    is_positive(is_positive)
{
}

std::unique_ptr<IMutableSentence> MutableQuantified::clone() const
{
    return std::make_unique<MutableQuantified>(
            quantifier_type, bound_term->clone(), sentence->clone(), is_positive);
}

void MutableQuantified::flip_polarity() noexcept
{
    is_positive = !is_positive;
}

bool MutableQuantified::is_negative_polarity() const noexcept
{
    return !is_positive;
}

void MutableQuantified::set_quantifier_type(const QuantifierTypes quantifier_type) noexcept
{
    this->quantifier_type = quantifier_type;
}

QuantifierTypes MutableQuantified::get_quantifier_type() const noexcept
{
    return quantifier_type;
}

const IMutableTerm *MutableQuantified::observe_bound_term() const noexcept
{
    return bound_term.get();
}

std::unique_ptr<IMutableTerm> MutableQuantified::take_bound_term() noexcept
{
    return std::move(bound_term);
}

std::unique_ptr<IMutableSentence> MutableQuantified::take_sentence() noexcept
{
    return std::move(sentence);
}

const IMutableSentence *MutableQuantified::observe_sentence() const noexcept
{
    return sentence.get();
}

void MutableQuantified::put_sentence(std::unique_ptr<IMutableSentence> &&sentence) noexcept
{
```

```
    this->sentence = std::move(sentence);
}

void MutableQuantified::accept(MutatingSentenceVisitorBase &visitor)
{
    visitor.visit(*this);
}

void MutableQuantified::accept(IObservingNodeVisitor &visitor) const
{
    visitor.visit(*this);
}

const IProcessedSentence *MutableQuantified::accept(RepositoryBuildingVisitor &visitor)
{
    return visitor.visit(*this);
}

std::size_t MutableQuantified::hash() const noexcept
{
    auto hash = std::hash<std::underlying_type_t<QuantifierTypes>>{}(std::to_underlying(quantifier_type));

    hash = hash_combine(hash, bound_term->hash());
    hash = hash_combine(hash, sentence->hash());

    return hash_polarity(hash, !is_positive);
}

std::ostream &MutableQuantified::serialise(std::ostream &ostream) const
{
    if (!is_positive)
        ostream << '~';

    ostream << '(' << get_operator_symbol(quantifier_type);
    bound_term->serialise(ostream);
    ostream << '(';
    sentence->serialise(ostream);
    return ostream << ')';
}

bool MutableQuantified::operator==(const IMutableSentence &other) const noexcept
{
    const auto other_quantified = dynamic_cast<const MutableQuantified *>(&other);
    if (other_quantified == nullptr)
        // Other IProcessedSentence isn't a MutableQuantified.
        return false;

    if (is_positive != other_quantified->is_positive)
        // Different signs.
        return false;

    if (quantifier_type != other_quantified->quantifier_type)
        // Different quantifier nature.
        return false;

    bool bound_terms_match = false;

    if (bound_term == nullptr)
        bound_terms_match = other_quantified->bound_term == nullptr;
    else if (other_quantified->bound_term == nullptr)
        bound_terms_match = bound_term == nullptr;
    else
        // Both bound terms are non-NULL and should be compared with IMutableTerm::operator==.
        bound_terms_match = *bound_term == *other_quantified->bound_term;

    if (!bound_terms_match)
        return false;

    /*
     * Bound terms and quantifier natures match. Test the sentences for NULL, otherwise use
     * IMutableSentence::operator==.
     */
    if (sentence == nullptr || other_quantified->sentence == nullptr)
        return sentence == other_quantified->sentence;

    return *sentence == *other_quantified->sentence;
}

void MutableQuantified::put_bound_term(std::unique_ptr<IMutableTerm> &&new_bound_term) noexcept
{
    bound_term = std::move(new_bound_term);
```

```
}

const char *MutableQuantified::get_operator_symbol(const QuantifierTypes type)
{
    switch (type) {
    case QuantifierTypes::Universal:
        return "ForAll ";
    case QuantifierTypes::Existential:
        return "ThereExists ";
    }

    return " ??? ";
}

} // namespace optifol
```

1.11.3.2.7 MutableQuantified.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the mutable Quantified Sentence IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#ifndef MUTABLEQUANTIFIED_HPP
#define MUTABLEQUANTIFIED_HPP

#include <memory>

#include "../../MutableVariants/Terms/IMutableTerm.hpp"
#include "../OwningBuildable.hpp"
#include "IMutableSentence.hpp"

namespace optifol
{

/**
 * @enum QuantifierTypes
 * @brief The modes by which a sentence can be bound by a variable
 */
enum class QuantifierTypes
{
    Universal,
    Existential
};

/**
 * @class MutableQuantified
 * @brief A MutableQuantified IR node owns a bound variable/term and a bound sentence ("operands"), and
 * detains the first-order logic operator by which the sentence is bound by the variable. Metadata and
 * operands are mutable, and owned by the instance of the IR node.
 * @note There is no immutable, non-owning dual of the MutableQuantified node as it may not appear in
 * CNF-normalised expression trees.
 */
class MutableQuantified : public IMutableSentence,
                          public OwningBuildable<MutableQuantified>
{
public:
    [[maybe_unused]] MutableQuantified(QuantifierTypes quantifier_type,
            std::unique_ptr<IMutableTerm> &&bound_term, std::unique_ptr<IMutableSentence> &&sentence,
            bool is_positive = true);

    [[nodiscard]] std::unique_ptr<IMutableSentence> clone() const override;

    void flip_polarity() noexcept override;

    [[nodiscard]] bool is_negative_polarity() const noexcept override;

    void accept(MutatingSentenceVisitorBase &visitor) override;

    void accept(IObservingNodeVisitor &visitor) const override;
```

```
    const IProcessedSentence *accept(RepositoryBuildingVisitor &visitor) override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] bool operator==(const IMutableSentence &other) const noexcept override;

    /**
     * @brief Replaces the quantifier operator with an alternative type
     * @param quantifier_type The new quantifier operator
     */
    void set_quantifier_type(QuantifierTypes quantifier_type) noexcept;

    /**
     * @brief Retrieve the quantification operator that binds the variable and the sentence
     * @return The quantifier type
     */
    [[nodiscard]] QuantifierTypes get_quantifier_type() const noexcept;

    /**
     * @brief Retrieves a pointer to the immutable bound term
     * @return An immutable pointer to the bound term
     * @see @ref std::unique_ptr::get for semantics of the observing getter
     */
    [[nodiscard]] const IMutableTerm *observe_bound_term() const noexcept;

    /**
     * @brief Steals ownership of the bound term from the object to the caller
     * @return The stolen container containing the bound term
     * @warning This call transfers ownership outbound
     */
    [[nodiscard]] std::unique_ptr<IMutableTerm> take_bound_term() noexcept;

    /**
     * @brief Steals ownership of the bound sentence from the object to the caller
     * @return The stolen container containing the bound sentence
     * @warning This call transfers ownership outbound
     */
    [[nodiscard]] std::unique_ptr<IMutableSentence> take_sentence() noexcept;

    /**
     * @brief Retrieves a pointer to the immutable bound sentence
     * @return An immutable pointer to the bound sentence
     * @see @ref std::unique_ptr::get for semantics of the observing getter
     */
    [[nodiscard]] const IMutableSentence *observe_sentence() const noexcept;

    /**
     * @brief Transfers ownership of a new bound sentence, overwriting any previously bound sentence
     * @param sentence The new sentence-owning container
     * @warning This call transfers ownership inbound
     * @see @ref std::unique_ptr::operator= for semantics of swap
     */
    void put_sentence(std::unique_ptr<IMutableSentence> &&sentence) noexcept;

    /**
     * @brief Transfers ownership of a new bound term, overwriting any previously bound term
     * @param new_bound_term The new term-owning container
     * @warning This call transfers ownership inbound
     * @see @ref std::unique_ptr::operator= for semantics of swap
     */
    void put_bound_term(std::unique_ptr<IMutableTerm> &&new_bound_term) noexcept;

    /**
     * @brief Maps a quantifier type to a human-readable string suitable for serialisation
     * @param type Type of quantifier to serialise
     * @return Serialised string for the given quantifier type
     */
    [[nodiscard]] static const char *get_operator_symbol(QuantifierTypes type);

private:
    QuantifierTypes quantifier_type;
    std::unique_ptr<IMutableTerm> bound_term;
    std::unique_ptr<IMutableSentence> sentence;
    bool is_positive;
};

} // namespace optifol

#endif
```

1.11.3.2.8 MutableSentenceRoot.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the mutable Sentence Root IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#include "MutableSentenceRoot.hpp"

#include "../../../Visitors/MutableTargets/Observers/IObservingNodeVisitor.hpp"
#include "../../../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../../../Visitors/MutableTargets/Sentences/MutatingSentenceVisitorBase.hpp"

namespace optifol
{

MutableSentenceRoot::MutableSentenceRoot(std::unique_ptr<IMutableSentence> &&sentence) :
    sentence(std::move(sentence))
{
}

std::unique_ptr<IMutableSentence> MutableSentenceRoot::clone() const
{
    return clone_as_root();
}

std::unique_ptr<MutableSentenceRoot> MutableSentenceRoot::clone_as_root() const
{
    return std::make_unique<MutableSentenceRoot>(sentence->clone());
}

void MutableSentenceRoot::flip_polarity() noexcept
{
    is_positive = !is_positive;
}

bool MutableSentenceRoot::is_negative_polarity() const noexcept
{
    return !is_positive;
}

std::unique_ptr<IMutableSentence> MutableSentenceRoot::take_sentence() noexcept
{
    return std::move(sentence);
}

const IMutableSentence *MutableSentenceRoot::observe_sentence() const noexcept
{
    return sentence.get();
}

void MutableSentenceRoot::put_sentence(std::unique_ptr<IMutableSentence> &&sentence) noexcept
{
    this->sentence = std::move(sentence);
}

bool MutableSentenceRoot::operator==(const IMutableSentence &other) const noexcept
{
    const auto other_root = dynamic_cast<const MutableSentenceRoot *>(&other);
    if (other_root == nullptr)
        // Other IMutableSentence isn't a MutableSentenceRoot.
        return false;

    if (is_positive != other_root->is_positive)
        // Different signs.
        return false;

    if (sentence == nullptr || other_root->sentence == nullptr)
        return sentence == other_root->sentence;

    return *sentence == *other_root->sentence;
}
```

```cpp
void MutableSentenceRoot::accept(MutatingSentenceVisitorBase &visitor)
{
    visitor.visit(*this);
}

void MutableSentenceRoot::accept(IObservingNodeVisitor &visitor) const
{
    visitor.visit(*this);
}

const IProcessedSentence *MutableSentenceRoot::accept(RepositoryBuildingVisitor &visitor)
{
    visitor.visit(*this);
    return nullptr;
}

std::size_t MutableSentenceRoot::hash() const noexcept
{
    return hash_polarity(sentence->hash(), is_negative_polarity());
}

std::ostream &MutableSentenceRoot::serialise(std::ostream &ostream) const
{
    if (is_negative_polarity())
        ostream << '~';

    ostream << '(';
    sentence->serialise(ostream);
    return ostream << ')';
}

} // namespace optifol
```

1.11.3.2.9 MutableSentenceRoot.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the mutable Sentence Root IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#ifndef MUTABLESENTENCEROOT_HPP
#define MUTABLESENTENCEROOT_HPP

#include "../OwningBuildable.hpp"
#include "IMutableSentence.hpp"

namespace optifol
{

/**
 * @class MutableSentenceRoot
 * @brief A MutableSentenceRoot denotes the root node of a mutable IR node tree. It owns a single sub-sentence
 * that may be mutated and transferred.
 */
class MutableSentenceRoot : public IMutableSentence,
                            public OwningBuildable<MutableSentenceRoot>
{
public:
    /**
     * @brief Construct a new SentenceRoot to encapsulate and own the given sub-sentence
     * @param sentence The container of the sentence whose ownership is to be transferred into the
     * SentenceRoot
     */
    [[maybe_unused]] explicit MutableSentenceRoot(std::unique_ptr<IMutableSentence> &&sentence);

    [[nodiscard]] std::unique_ptr<IMutableSentence> clone() const override;

    /**
     * @brief Recursively clone the MutableSentenceRoot and preserve the root typing.
     * @return The container containing the copied sentence root.
```

```
     */
    [[nodiscard]] std::unique_ptr<MutableSentenceRoot> clone_as_root() const;

    void flip_polarity() noexcept override;

    [[nodiscard]] bool is_negative_polarity() const noexcept override;

    void accept(MutatingSentenceVisitorBase &visitor) override;

    void accept(IObservingNodeVisitor &visitor) const override;

    /**
     * @copydetails IMutableSentence::accept(RepositoryBuildingVisitor&)
     * @return A @ref std::nullptr_t value, as roots are not managed by any SymbolRepository.
     */
    const IProcessedSentence *accept(RepositoryBuildingVisitor &visitor) override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    std::ostream &serialise(std::ostream &ostream) const override;

    /**
     * @brief Steals ownership of the contained sentence
     * @return The stolen container of the sentence
     * @warning This call transfers ownership outbound
     */
    [[nodiscard]] std::unique_ptr<IMutableSentence> take_sentence() noexcept;

    /**
     * @brief Retrieves a pointer to the immutable owned sentence
     * @return An immutable pointer to the owned sentence
     */
    [[nodiscard]] const IMutableSentence *observe_sentence() const noexcept;

    /**
     * @brief Transfers ownership of a new sentence, overwriting any previously held sentence
     * @param sentence The new sentence-owning container
     * @warning This call transfers ownership inbound
     * @see @ref std::unique_ptr::operator= for semantics of swap
     */
    void put_sentence(std::unique_ptr<IMutableSentence> &&sentence) noexcept;

    [[nodiscard]] bool operator==(const IMutableSentence &other) const noexcept override;

private:
    std::unique_ptr<IMutableSentence> sentence;
    bool is_positive = true;
};

} // namespace optifol

#endif
```

1.11.3.3 Terms

1.11.3.3.1 IMutableTerm.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the generic mutable IR Term interface
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#ifndef IMUTABLETERM_HPP
#define IMUTABLETERM_HPP

#include <memory>

#include "../../Terms/ITerm.hpp"

namespace optifol
{
```

```
class MutatingTermVisitorBase;
class RepositoryBuildingVisitor;
class IProcessedTerm;
class IObservingNodeVisitor;

/**
 * @class IMutableTerm
 * @brief An IMutableTerm is an ITerm that has not undergone the full lexing, parsing, and normalisation
 * pipeline. Such terms are still being processed and likely to be mutated in-situ.
 * @details IMutableTerms are typically held within an outermost @ref std::unique_ptr to enforce clear
 * semantics of ownership, and the explicit transfer thereof, throughout the mutation pipelines. Once an
 * IMutableTerm has been deemed as processed, with no further mutations necessary, it should be converted to
 * an IProcessedTerm that has more restrictions but does not require equally stringent ownership.
 */
class IMutableTerm : public ITerm
{

public:
    /**
     * @brief Recursively clone an owning term, making replicas of all children held by the cloned parent, and
     * return the root-most node detained by a @ref std::unique_ptr.
     * @return The transferable container holding the recursively cloned term
     */
    [[nodiscard]] virtual std::unique_ptr<IMutableTerm> clone() const = 0;

    /**
     * @brief Accept a visitation from a MutatingTermVisitorBase-type visitor
     * @param visitor The instantiation of the mutating term visitor
     */
    virtual void accept(MutatingTermVisitorBase &visitor) = 0;

    /**
     * @brief Accept a visitation from an observing (non-mutating) term visitor on the object
     * @param visitor The observing visitor to invoke
     */
    virtual void accept(IObservingNodeVisitor &visitor) const = 0;

    /**
     * @brief Accept a visitation from a RepositoryBuildingVisitor-type visitor
     * @param visitor The instantiation of the mutating repository-building visitor
     */
    [[nodiscard]] virtual const IProcessedTerm *accept(RepositoryBuildingVisitor &visitor) = 0;

    /**
     * @brief Test equality between two IMutableTerm instances.
     * @param other The other IMutableTerm instance with which to test for equality.
     * @return Are the two terms of the same concrete type and equal?
     */
    [[nodiscard]] virtual bool operator==(const IMutableTerm &other) const noexcept = 0;
};

} // namespace optifol

#endif
```

1.11.3.3.2 MutableFunction.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the mutable Function IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#include "MutableFunction.hpp"

#include "../../../CompositeSerialisationHelpers.hpp"
#include "../../../Visitors/MutableTargets/Observers/IObservingNodeVisitor.hpp"
#include "../../../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../../../Visitors/MutableTargets/Terms/MutatingTermVisitorBase.hpp"
#include "../../Terms/Function.hpp"
```

```cpp
namespace optifol
{

MutableFunction::MutableFunction(std::string name, std::vector<std::unique_ptr<IMutableTerm>> &&arguments) :
    name(std::move(name)),
    arguments(std::move(arguments))
{
}

std::string MutableFunction::to_string() const
{
    return CompositeSerialisationHelpers::string_serialise(name, arguments.cbegin(), arguments.cend());
}

std::unique_ptr<IMutableTerm> MutableFunction::clone() const
{
    std::vector<std::unique_ptr<IMutableTerm>> cloned_arguments;
    cloned_arguments.reserve(arguments.size());
    for (const auto &argument: arguments)
        cloned_arguments.push_back(argument->clone());

    return std::make_unique<MutableFunction>(name, std::move(cloned_arguments));
}

std::string_view MutableFunction::get_disambiguated_name() const
{
    return name;
}

void MutableFunction::accept(MutatingTermVisitorBase &visitor)
{
    visitor.visit(*this);
}

const IProcessedTerm *MutableFunction::accept(RepositoryBuildingVisitor &visitor)
{
    return visitor.visit(*this);
}

void MutableFunction::accept(IObservingNodeVisitor &visitor) const
{
    visitor.visit(*this);
}

bool MutableFunction::operator==(const IMutableTerm &other) const noexcept
{
    const auto other_function = dynamic_cast<const MutableFunction *>(&other);
    if (other_function == nullptr)
        // Other IMutableTerm isn't a MutableFunction.
        return false;

    if (get_disambiguated_name() != other_function->get_disambiguated_name())
        // Different superficial names.
        return false;

    const auto argument_count = arguments.size();
    if (argument_count != other_function->arguments.size())
        // Different number of arguments.
        return false;

    for (std::size_t argument_idx = 0; argument_idx < argument_count; ++argument_idx) {
        const auto &lhs_arg_ptr = arguments[argument_idx];
        const auto &rhs_arg_ptr = other_function->arguments[argument_idx];

        if (lhs_arg_ptr == nullptr) {
            if (rhs_arg_ptr != nullptr)
                return false;
        } else if (rhs_arg_ptr == nullptr)
            return false;
        else if (*lhs_arg_ptr != *rhs_arg_ptr)
            return false;
    }

    return true;
}

const std::vector<std::unique_ptr<IMutableTerm>> &MutableFunction::observe_arguments() const noexcept
{
    return arguments;
}
```

```
std::vector<std::unique_ptr<IMutableTerm>> &MutableFunction::observe_arguments() noexcept
{
    return arguments;
}

std::ostream &MutableFunction::serialise(std::ostream &ostream) const
{
    return CompositeSerialisationHelpers::stream_serialise(
            ostream, name, arguments.cbegin(), arguments.cend());
}

std::size_t MutableFunction::hash() const noexcept
{
    return composite_hash(name, arguments.cbegin(), arguments.cend());
}

} // namespace optifol
```

1.11.3.3.3 MutableFunction.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the mutable Function IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#ifndef MUTABLEFUNCTION_HPP
#define MUTABLEFUNCTION_HPP

#include <memory>
#include <vector>

#include "../OwningBuildable.hpp"
#include "IMutableTerm.hpp"

namespace optifol
{

/**
 * @class MutableFunction
 * @brief A MutableFunction represents an owning IR node term consisting of a display name and zero to many
 * arguments, all of which are owned as transferable @ref std::unique_ptr objects by the node.
 * @see Function for the processed, argument-observing dual.
 */
class MutableFunction : public IMutableTerm,
                        public OwningBuildable<MutableFunction>
{
public:
    /**
     * @brief Create an owning unsigned function with an initial set of owned arguments.
     * @param name Display name of the function
     * @param arguments A moveable ordered container owning the initial arguments
     */
    [[maybe_unused]] explicit MutableFunction(
            std::string name, std::vector<std::unique_ptr<IMutableTerm>> &&arguments = {});

    [[nodiscard]] std::string to_string() const override;

    [[nodiscard]] std::unique_ptr<IMutableTerm> clone() const override;

    [[nodiscard]] std::string_view get_disambiguated_name() const override;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    void accept(MutatingTermVisitorBase &visitor) override;

    const IProcessedTerm *accept(RepositoryBuildingVisitor &visitor) override;

    void accept(IObservingNodeVisitor &visitor) const override;
```

```
    [[nodiscard]] bool operator==(const IMutableTerm &other) const noexcept override;

    /**
     * @brief Observe the constant owning ordered argument collection
     * @return The arguments owned by the function
     */
    [[nodiscard]] const std::vector<std::unique_ptr<IMutableTerm>> &observe_arguments() const noexcept;

    /**
     * @brief Observe the mutable owning ordered argument collection
     * @return The arguments owned by the function
     * @note This non-constant overload is useful for propagation of <code>accept</code> calls on mutating
     * visitors.
     */
    [[nodiscard]] std::vector<std::unique_ptr<IMutableTerm>> &observe_arguments() noexcept;

protected:
    std::string name;
    std::vector<std::unique_ptr<IMutableTerm>> arguments;
};

} // namespace optifol

#endif
```

1.11.3.3.4 MutableSkolemFunction.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the mutable Skolem Function Term IR node
 * @author Oliver Dixon
 * @date 2024-11-30
 * @version Development
 */

#include "MutableSkolemFunction.hpp"

#include "../../../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../../Terms/SkolemFunction.hpp"

namespace optifol
{

MutableSkolemFunction::MutableSkolemFunction(
        std::string name, std::vector<std::unique_ptr<IMutableTerm>> &&quantified_variables) :
    MutableFunction(std::move(name), std::move(quantified_variables))
{
}

std::unique_ptr<IMutableTerm> MutableSkolemFunction::clone() const
{
    std::vector<std::unique_ptr<IMutableTerm>> cloned_arguments;

    cloned_arguments.reserve(arguments.size());
    for (const auto &argument: arguments)
        cloned_arguments.push_back(argument->clone());

    return std::make_unique<MutableSkolemFunction>(name, std::move(cloned_arguments));
}

const IProcessedTerm *MutableSkolemFunction::accept(RepositoryBuildingVisitor &visitor)
{
    return visitor.visit(*this);
}

} // namespace optifol
```

1.11.3.3.5 MutableSkolemFunction.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
```

```
 */

/**
 * @file
 * @brief Class specification for the mutable Skolem Function Term IR node
 * @author Oliver Dixon
 * @date 2024-11-30
 * @version Development
 */

#ifndef MUTABLESKOLEMFUNCTION_HPP
#define MUTABLESKOLEMFUNCTION_HPP

#include <vector>

#include "MutableFunction.hpp"

namespace optifol
{

class MutableVariable;

class MutableSkolemFunction : public MutableFunction
{
public:
    [[maybe_unused]] explicit MutableSkolemFunction(
            std::string name, std::vector<std::unique_ptr<IMutableTerm>> &&quantified_variables = {});

    [[nodiscard]] std::unique_ptr<IMutableTerm> clone() const override;

    const IProcessedTerm *accept(RepositoryBuildingVisitor &visitor) override;
};

} // namespace optifol

#endif
```

1.11.3.3.6 MutableVariable.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the mutable Variable Term IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#include "MutableVariable.hpp"

#include "../../../Visitors/MutableTargets/Observers/IObservingNodeVisitor.hpp"
#include "../../../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../../../Visitors/MutableTargets/Terms/MutatingTermVisitorBase.hpp"
#include "../../Terms/Variable.hpp"
#include "MutableFunction.hpp"

namespace optifol
{

MutableVariable::MutableVariable(std::string name) :
    name(std::move(name))
{
}

MutableVariable::MutableVariable(std::string name, const std::string &disambiguated_name) :
    name(std::move(name)),
    disambiguated_name(disambiguated_name)
{
}

std::unique_ptr<IMutableTerm> MutableVariable::clone() const
{
    if (disambiguated_name.has_value())
        return std::make_unique<MutableVariable>(name, *disambiguated_name);
```

```
    return std::make_unique<MutableVariable>(name);
}

void MutableVariable::accept(MutatingTermVisitorBase &visitor)
{
    visitor.visit(*this);
}

std::string MutableVariable::to_string() const
{
    return std::string(get_disambiguated_name());
}

std::string_view MutableVariable::get_disambiguated_name() const
{
    if (disambiguated_name.has_value())
        return *disambiguated_name;

    return name;
}

std::string_view MutableVariable::get_base_name() const noexcept
{
    return name;
}

const IProcessedTerm *MutableVariable::accept(RepositoryBuildingVisitor &visitor)
{
    return visitor.visit(*this);
}

void MutableVariable::accept(IObservingNodeVisitor &visitor) const
{
    visitor.visit(*this);
}

bool MutableVariable::operator==(const IMutableTerm &other) const noexcept
{
    const auto other_variable = dynamic_cast<const MutableVariable *>(&other);
    if (other_variable == nullptr)
        // Other IMutableTerm isn't a MutableVariable.
        return false;

    return get_disambiguated_name() == other_variable->get_disambiguated_name();
}

} // namespace optifol
```

1.11.3.3.7 MutableVariable.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the mutable Variable Term IR node
 * @author Oliver Dixon
 * @date 2025-05-06
 * @version Development
 */

#ifndef MUTABLEVARIABLE_HPP
#define MUTABLEVARIABLE_HPP

#include <optional>

#include "../OwningBuildable.hpp"
#include "IMutableTerm.hpp"

namespace optifol
{

/**
 * @class MutableVariable
 * @brief A MutableVariable is an owning IR node representing a first-order logic non-free variable. Within
 * the semantics of Optifol, it must be used in the context of a MutableQuantified IR node. Variables have
 * display names and disambiguated names, which may or may not be identical, although the disambiguated name
```

```
 * is unique in the scope of the outermost sentence.
 * @see Variable for the non-owning, immutable dual
 */
class MutableVariable : public IMutableTerm,
                        public OwningBuildable<MutableVariable>
{
public:
    /**
     * @brief Create a new MutableVariable with an initial display name
     * @param name The initial name of the variable
     */
    explicit MutableVariable(std::string name);

    /**
     * @brief Create a new MutableVariable with an initial display name and disambiguated name
     * @param name The initial name of the variable
     * @param disambiguated_name The initial disambiguated name for the variable
     * @warning No uniqueness check is done for the disambiguated name upon construction
     */
    explicit MutableVariable(std::string name, const std::string &disambiguated_name);

    [[nodiscard]] std::unique_ptr<IMutableTerm> clone() const override;

    void accept(MutatingTermVisitorBase &visitor) override;

    [[nodiscard]] std::string to_string() const override;

    [[nodiscard]] std::string_view get_disambiguated_name() const override;

    [[nodiscard]] std::string_view get_base_name() const noexcept;

    const IProcessedTerm *accept(RepositoryBuildingVisitor &visitor) override;

    void accept(IObservingNodeVisitor &visitor) const override;

    [[nodiscard]] bool operator==(const IMutableTerm &other) const noexcept override;

    static constexpr char disambiguating_delimiter = '#';

private:
    std::string name;
    std::optional<std::string> disambiguated_name;
};

} // namespace optifol

#endif
```

1.11.4 Sentences

1.11.4.1 BinaryConnected.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the binary-Connected Sentence IR node
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#include <utility>

#include "../../Visitors/MutableTargets/Observers/LaTeXSerialisationVisitor.hpp"
#include "BinaryConnected.hpp"

namespace optifol
{

BinaryConnected::BinaryConnected(const BinaryOperatorTypes operator_type, const IProcessedSentence *const lhs,
        const IProcessedSentence *const rhs, const bool is_positive) :
    operator_type(operator_type),
    lhs(lhs),
    rhs(rhs),
    is_positive(is_positive)
```

```
{
}

bool BinaryConnected::is_negative_polarity() const noexcept
{
    return !is_positive;
}

std::size_t BinaryConnected::hash() const noexcept
{
    return hash_binary_connected(operator_type, lhs, rhs, is_negative_polarity());
}

std::ostream &BinaryConnected::serialise(std::ostream &ostream) const
{
    return serialise_binary_connected(ostream, operator_type, lhs, rhs, is_negative_polarity());
}

bool BinaryConnected::operator==(const IProcessedSentence &other) const noexcept
{
    const auto other_connected = dynamic_cast<const BinaryConnected *>(&other);
    if (other_connected == nullptr)
        // Other IProcessedSentence isn't a BinaryConnected.
        return false;

    return commutative_ptr_compare(lhs, rhs, other_connected->lhs, other_connected->rhs);
}

BinaryOperatorTypes BinaryConnected::get_operator_type() const noexcept
{
    return operator_type;
}

const IProcessedSentence *BinaryConnected::observe_lhs_operand() const noexcept
{
    return lhs;
}

const IProcessedSentence *BinaryConnected::observe_rhs_operand() const noexcept
{
    return rhs;
}

std::size_t BinaryConnected::hash_binary_connected(const BinaryOperatorTypes operator_type,
        const ISentence *const lhs, const ISentence *const rhs, const bool is_negative_polarity) noexcept
{
    return hash_polarity(hash_combine(hash_combine_commutative(lhs->hash(), rhs->hash()),
                                      std::hash<std::size_t>{}(std::to_underlying(operator_type))),
            is_negative_polarity);
}

std::ostream &BinaryConnected::serialise_binary_connected(std::ostream &ostream,
        const BinaryOperatorTypes operator_type, const ISentence *const lhs, const ISentence *const rhs,
        const bool is_negative_polarity)
{
    if (is_negative_polarity)
        ostream << '~';

    ostream << '(';
    lhs->serialise(ostream);
    ostream << get_operator_symbol(operator_type);
    return rhs->serialise(ostream) << ')';
}

const char *BinaryConnected::get_operator_symbol(const BinaryOperatorTypes type)
{
    switch (type) {
    case BinaryOperatorTypes::Conjunction:
        return " & ";
    case BinaryOperatorTypes::Disjunction:
        return " | ";
    case BinaryOperatorTypes::Implication:
        return " => ";
    case BinaryOperatorTypes::Biconditional:
        return " <=> ";
    }

    return " ??? ";
}

} // namespace optifol
```

1.11.4.2 BinaryConnected.hpp

```hpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the binary-Connected Sentence IR node
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#ifndef BINARYCONNECTED_HPP
#define BINARYCONNECTED_HPP

#include "IProcessedSentence.hpp"

namespace optifol
{

/**
 * @enum BinaryOperatorTypes
 * @brief Types of operators usable by binary-connected sentences to connect two operands
 */
enum class BinaryOperatorTypes
{
    Conjunction,
    Disjunction,
    Implication,
    Biconditional,
};

/**
 * @class BinaryConnected
 * @brief A BinaryConnected IR node references two operands semantically joined with an operator. Metadata and
 * operands are immutable; operands are held centrally in a SymbolRepository.
 * @see BinaryOperatorTypes for modes of connection
 * @see MutableBinaryConnected for the owning dual
 */
class BinaryConnected : public IProcessedSentence
{
public:
    /**
     * @brief Create a signed binary-connected IR node
     * @param operator_type Operator with which the operands are connected
     * @param lhs The left-hand operand reference in the SymbolRepository
     * @param rhs The right-hand operand reference in the SymbolRepository
     * @param is_positive Should the node be instantiated with in a positive polarity?
     */
    explicit BinaryConnected(BinaryOperatorTypes operator_type, const IProcessedSentence *lhs,
            const IProcessedSentence *rhs, bool is_positive = true);

    [[nodiscard]] bool is_negative_polarity() const noexcept override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] bool operator==(const IProcessedSentence &other) const noexcept override;

    /**
     * @brief Get the fixed operator by which the operands are connected
     * @return The binary-connected operator type
     */
    [[nodiscard]] BinaryOperatorTypes get_operator_type() const noexcept;

    /**
     * @brief Provides an immutable observing pointer to the owned LHS operand
     * @return The LHS operand observer
     */
    [[nodiscard]] const IProcessedSentence *observe_lhs_operand() const noexcept;

    /**
     * @brief Provides an immutable observing pointer to the owned RHS operand
     * @return The RHS operand observer
     */
    [[nodiscard]] const IProcessedSentence *observe_rhs_operand() const noexcept;
```

```
    /**
     * @brief Hash any type of binary-connected IR node from a static context
     * @param operator_type The type of binary operator with which the operands are connected
     * @param lhs An observing pointer to the LHS operand
     * @param rhs An observing pointer to the RHS operand
     * @param is_negative_polarity Has the target been instantiated with a negative polarity?
     * @return Numeric hash of the binary-connected node described by the given parameters
     */
    [[nodiscard]] static std::size_t hash_binary_connected(BinaryOperatorTypes operator_type,
            const ISentence *lhs, const ISentence *rhs, bool is_negative_polarity) noexcept;

    /**
     * @brief Serialise any type of binary-connected IR node from a static context into an output stream
     * @param ostream Destination output stream
     * @param operator_type The type of binary operator with which the operands are connected
     * @param lhs An observing pointer to the LHS operand
     * @param rhs An observing pointer to the RHS operand
     * @param is_negative_polarity Has the target been instantiated with a negative polarity?
     * @return Populated destination output stream
     */
    static std::ostream &serialise_binary_connected(std::ostream &ostream, BinaryOperatorTypes operator_type,
            const ISentence *lhs, const ISentence *rhs, bool is_negative_polarity);

    /**
     * @brief Return the appropriate human-readable symbol for a BinaryOperatorTypes value
     * @param type The serialisation target
     * @return A human-readable string representation of a binary operator
     */
    [[nodiscard]] static const char *get_operator_symbol(BinaryOperatorTypes type);

private:
    const BinaryOperatorTypes operator_type;

    const IProcessedSentence *const lhs;

    const IProcessedSentence *const rhs;

    const bool is_positive;
};

} // namespace optifol

#endif
```

1.11.4.3 Clause.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Clause collection
 * @author Oliver Dixon
 * @date 2025-09-15
 * @version Development
 */

#include "Clause.hpp"

#include <algorithm>
#include <ranges>

#include "../Inference/Resolvent.hpp"
#include "../Visitors/RegularTargets/FeatureBuildingVisitor.hpp"
#include "../Visitors/RegularTargets/Unification/UnificationVisitor.hpp"

namespace optifol
{

Clause::Clause(const std::initializer_list<const Literal *> literals) :
    literals(literals)
{
    if (!this->literals.empty()) {
        std::ranges::sort(this->literals,
                [](const Literal *const lhs, const Literal *const rhs) { return *lhs < *rhs; });
```

```
        if (is_tautology()) {
            this->literals.clear();
            state = State::TriviallyTrue;
        } else
            state = State::NotTrivial;
    }

    recompute_feature_vector();
}

void Clause::add_literal(const Literal *const new_literal)
{
    const auto nearest_lower = std::ranges::lower_bound(literals, new_literal,
            [](const Literal *const lhs, const Literal *const rhs) { return *lhs < *rhs; });

    /*
     * The nearest lower element is already either identical to the new literal, or its complement. In the
     * former case, we can omit adding it entirely; in the latter case, the entire clause becomes trivial.
     */
    if (nearest_lower != literals.cend() && (*nearest_lower)->unsigned_equality(*new_literal)) {
        if ((*nearest_lower)->is_negative_polarity() != new_literal->is_negative_polarity()) {
            literals.clear();
            state = State::TriviallyTrue;
        }

        return;
    }

    // Do a similar check just prior to the insertion point, if this could identify a tautology.
    if (nearest_lower != literals.cbegin()) {
        const auto previous_literal = std::prev(nearest_lower);
        if ((*previous_literal)->unsigned_equality(*new_literal)) {
            if ((*previous_literal)->is_negative_polarity() != new_literal->is_negative_polarity()) {
                literals.clear();
                state = State::TriviallyTrue;
            }

            return;
        }
    }

    // If this is a new unseen literal, insert at the correct position to maintain ordering i.a.w.
    // Literal::operator<.
    literals.insert(nearest_lower, new_literal);
    state = State::NotTrivial;
    recompute_feature_vector(*new_literal);
}

Clause::State Clause::get_triviality_state() const noexcept
{
    return state;
}

decltype(Clause::literals)::const_iterator Clause::begin() const noexcept
{
    return literals.cbegin();
}

decltype(Clause::literals)::const_iterator Clause::end() const noexcept
{
    return literals.cend();
}

std::ostream &Clause::serialise(std::ostream &ostream) const
{
    ostream << '{' << ' ';

    if (!literals.empty()) {
        std::ranges::for_each_n(literals.begin(),
                literals.size() - 1, // NOLINT(*-narrowing-conversions)
                [&ostream](const Literal *literal) { ostream << *literal << ',' << ' '; });

        ostream << *literals.back();
    }

    return ostream << ' ' << '}';
}

std::size_t Clause::hash() const noexcept
{
    return std::ranges::fold_left(literals, std::size_t{0},
```

```
            [](const std::size_t seed, const Literal *const literal)
            { return hash_combine(seed, std::hash<Literal>{}(*literal)); });
}

bool Clause::operator<(const Clause &other) const noexcept
{
    return std::ranges::lexicographical_compare(
            literals, other.literals, [](const Literal *lhs, const Literal *rhs) { return *lhs < *rhs; });
}

bool Clause::operator==(const Clause &other) const noexcept
{
    return std::ranges::equal(literals, other.literals,
            [](const Literal *lhs, const Literal *rhs) { return lhs->operator==(*rhs); });
}

std::size_t Clause::order() const noexcept
{
    return literals.size();
}

bool Clause::empty() const noexcept
{
    return literals.empty();
}

bool Clause::is_unit() const noexcept
{
    return order() == 1;
}

const Clause *Clause::observe_node() const noexcept
{
    return this;
}

std::optional<const Unifier *> Clause::observe_edge() const noexcept
{
    return std::nullopt;
}

const std::vector<Feature> &Clause::observe_features() const noexcept
{
    return features;
}

void Clause::accept(FeatureBuildingVisitor &feature_component_builder) const
{
    feature_component_builder.visit(this);
}

bool Clause::subsumes(const Clause &other_clause, UnificationVisitor &visitor) const
{
    if (empty())
        return false;

    for (const auto literal: *this) {
        const auto can_be_unified = [&visitor, lhs = literal](const Literal *const rhs)
        { return lhs->accept(visitor, *rhs); };

        if (!std::ranges::any_of(other_clause.literals, can_be_unified))
            return false;
    }

    return true;
}

bool Clause::is_tautology() const noexcept
{
    return std::ranges::adjacent_find(literals,
                [](const Literal *lhs, const Literal *rhs)
                {
                    return lhs->unsigned_equality(*rhs) &&
                            lhs->is_negative_polarity() == !rhs->is_negative_polarity();
                }) != literals.end();
}

void Clause::recompute_feature_vector() noexcept
{
    accept(feature_building_visitor);
    features = feature_building_visitor.extract_sorted_vector();
```

```
}

void Clause::recompute_feature_vector(const Literal &literal) noexcept
{
    literal.accept(feature_building_visitor);
    auto new_features = feature_building_visitor.extract_sorted_vector();

    if (new_features.size() > features.size())
        features = std::move(new_features);
    else
        for (const auto &new_feature: new_features)
            Feature::get(features, new_feature.get_feature_type()).join(new_feature);
}

} // namespace optifol
```

1.11.4.4 Clause.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Clause collection
 * @author Oliver Dixon
 * @date 2025-09-15
 * @version Development
 */

#ifndef OPTIFOL_CLAUSE_HPP
#define OPTIFOL_CLAUSE_HPP

#include <vector>

#include "../../Inference/ProofTreeNode.hpp"
#include "../Visitors/RegularTargets/FeatureBuildingVisitor.hpp"
#include "Literal.hpp"

namespace optifol
{

class UnificationVisitor;
class Literal;

/**
 * @class Clause
 * @brief A Clause is an unordered set of Literal nodes considered under disjunction.
 * @details
 *  <p>
 *      An @f$ N @f$-Clause @f$ C @f$ is represented in FOL as
 *      @f[ C = \left\{ L_1, \ldots, L_N \right\} = L_1 \lor \ldots \lor L_N @f] for Literal nodes
 *      @f$ L_1, \ldots, L_N @f$. Under standard FOL rules, a Clause is trivially false ("bottom") if
 *      @f$ C = \emptyset @f$, and likewise trivially true ("a tautology") if
 *      @f$ \exists L_i \in C \left( \lnot L_i \in C \right) @f$.
 *  </p>
 *  <p>
 *      This implementation models a tautology with @f$ C := \emptyset @f$. Trivial states can be further
 * queried.
 *  </p>
 */
class Clause : public OwningBuildable<Clause>,
               public IHashable,
               public ProofTreeNode
{
public:
    /**
     * @enum State
     * @brief Triviality state of the Clause, indicating a bottom (unsatisfiable) or tautology (trivially
     * satisfiable).
     */
    enum class State
    {
        NotTrivial,
        TriviallyTrue,
        TriviallyFalse
    };
```

```
private:
    /**
     * @brief The collection of Literal objects
     * @note An ordering according to Literal::operator<(const Literal&) is currently enforced on this
     * container by
     *  @ref Clause::Clause() and Clause::add_literal. Mathematically, a Clause is unordered due to the
     * associativity of FOL disjunction, but it's easier to keep an ordering here to detect tautologies or
     * duplicate entries.
     */
    std::vector<const Literal *> literals;

    State state = State::TriviallyFalse;

public:
    /**
     * @brief Construct a new trivially false Clause.
     */
    Clause() = default;

    /**
     * @brief Construct a new Clause with an initial set of Literals
     * @param literals The initial set of Literals to be considered for addition to the Clause
     */
    explicit Clause(std::initializer_list<const Literal *> literals);

    /**
     * @brief Add a new Literal to the Clause store, subject to conditions.
     * @details Mutating the state of the Clause can change its structure. In particular:
     *  <ul>
     *      <li>If the same Literal is already present, it is silently rejected.</li>
     *      <li>If the complentary Literal is already present, the Clause is deemed a tautology and has its
     * store cleared and state updated accordingly.</li> <li>If none of the above, it is added. If the Clause
     * was previously bottom, it is updated to non-trivial.</li>
     *  </ul>
     * @param new_literal The incoming Literal to consider adding.
     */
    void add_literal(const Literal *new_literal);

    /**
     * @brief Reports on the triviality state of the Clause, indicating whether it is a tautology.
     * @return Triviality state of the Clause.
     */
    [[nodiscard]] State get_triviality_state() const noexcept;

    /**
     * @brief Retrieve a constant iterator to the beginning of the underlying Literal storage container.
     * @return An iterator to the first Literal in the Clause.
     */
    [[nodiscard]] decltype(literals)::const_iterator begin() const noexcept;

    /**
     * @brief Retrieve a constant iterator to one-past-the-end of the underlying Literal storage container.
     * @return An iterator to one-past-the-end Literal in the Clause.
     */
    [[nodiscard]] decltype(literals)::const_iterator end() const noexcept;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    bool operator<(const Clause &other) const noexcept;

    /**
     * @brief Determine equality between two Clauses by zipping the Literal nodes
     * @param other The other Clause to test
     * @return Are the two Clauses equal?
     */
    [[nodiscard]] bool operator==(const Clause &other) const noexcept;

    /**
     * @brief Gets the size/order of the Clause.
     * @return The number of Literal objects in the Clause.
     */
    [[nodiscard]] std::size_t order() const noexcept;

    /**
     * @brief Is the Clause empty?
     * @return Is the Clause empty, or equivalently, trivially false?
     */
    [[nodiscard]] bool empty() const noexcept;
```

```
    /**
     * @brief Is the Clause a unit clause?
     * @return Is the Clause a unit, or equivalently, of order 1?
     */
    [[nodiscard]] bool is_unit() const noexcept override;

    [[nodiscard]] const Clause *observe_node() const noexcept override;

    [[nodiscard]] std::optional<const Unifier *> observe_edge() const noexcept override;

    [[nodiscard]] const std::vector<Feature> &observe_features() const noexcept;

    void accept(FeatureBuildingVisitor &feature_component_builder) const;

    [[nodiscard]] bool subsumes(const Clause &other_clause, UnificationVisitor &visitor) const;

private:
    /**
     * @brief Checks whether there exists a complementary pair of Literal objects in the Clause.
     * @return Is the Clause a tautology?
     * @pre The Literal objects are sorted in the canonical manner; see @ref literals.
     */
    [[nodiscard]] bool is_tautology() const noexcept;

    void recompute_feature_vector() noexcept;

    void recompute_feature_vector(const Literal &literal) noexcept;

    std::vector<Feature> features;
    FeatureBuildingVisitor feature_building_visitor;
};

} // namespace optifol

#endif // OPTIFOL_CLAUSE_HPP
```

1.11.4.5 IProcessedSentence.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the IProcessedSentence interface
 * @author Oliver Dixon
 * @date 2025-06-07
 * @version Development
 */

#ifndef IPROCESSEDSENTENCE_HPP
#define IPROCESSEDSENTENCE_HPP

#include "ISentence.hpp"

namespace optifol
{

class IObservingBinaryVisitor;
class Literal;

/**
 * @class IProcessedSentence
 * @brief An IProcessedSentence is the interface for immutable sentence AST nodes that have been normalised
 * and are no longer editable. They're typically stored in a global SymbolRepository and referenced with
 * observing pointers.
 */
class IProcessedSentence : public ISentence
{
public:
    [[nodiscard]] virtual bool accept(
            IObservingBinaryVisitor &binary_visitor, const IProcessedSentence &sentence) const
    {
        std::ignore = binary_visitor;
        std::ignore = sentence;
        return false;
    }
```

```
    [[nodiscard]] virtual bool accept(
            IObservingBinaryVisitor &unification_visitor, const Literal &predicate) const
    {
        std::ignore = unification_visitor;
        std::ignore = predicate;
        return false;
    }

    /**
     * @brief Test equality between two IProcessedSentence instances.
     * @param other The other IProcessedSentence instance with which to test for equality.
     * @return Are the two sentences of the same concrete type and equal?
     */
    [[nodiscard]] virtual bool operator==(const IProcessedSentence &other) const noexcept = 0;
};

} // namespace optifol

#endif // IPROCESSEDSENTENCE_HPP
```

1.11.4.6 ISentence.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the generic IR Sentence interface
 * @author Oliver Dixon
 * @date 2025-06-14
 * @version Development
 */

#ifndef ISENTENCE_HPP
#define ISENTENCE_HPP

#include "../../IHashable.hpp"
#include "../../ISerialisable.hpp"
#include "../../Optifol.hpp"

namespace optifol
{

/**
 * @class ISentence
 * @brief A sentence represents a first-order logic sentence node within an IR tree. In most instantiations,
 * it may consist of a deeply nested structure and always supports a polarity, i.e. positive or negative.
 */
class ISentence : public IHashable,
                  public ISerialisable
{
public:
    [[nodiscard]] virtual bool is_negative_polarity() const noexcept = 0;

protected:
    /**
     * @brief Commutatively compare the LHS and RHS nodes of two objects for equality.
     * @tparam LHSType The type of the LHS nodes.
     * @tparam RHSType The type of the RHS nodes.
     * @param my_lhs The LHS of the first node.
     * @param my_rhs The RHS of the first node.
     * @param their_lhs The LHS of the second node.
     * @param their_rhs The RHS of the second node.
     * @return Does the first LHS equal the LHS or RHS of the second node, and does the first RHS equal the
     * LHS or RHS of the second node?
     */
    template<class LHSType, class RHSType>
        requires WeaklyEqualityComparableWith<LHSType, RHSType>
    [[nodiscard]] static bool commutative_ptr_compare(const LHSType *const my_lhs,
            const RHSType *const my_rhs, const LHSType *const their_lhs,
            const RHSType *const their_rhs) noexcept
    {
        // If the first equality check fails, try flipping the "us" arguments to match the order of "their"
        // arguments.
        return noncommutative_ptr_compare(my_lhs, my_rhs, their_lhs, their_rhs) ||
                noncommutative_ptr_compare(my_rhs, my_lhs, their_lhs, their_rhs);
    }
```

```
    /**
     * @brief Non-commutatively compare the LHS and RHS nodes of two objects for equality.
     * @tparam LHSType The type of the LHS nodes.
     * @tparam RHSType The type of the RHS nodes.
     * @param my_lhs The LHS of the first node.
     * @param my_rhs The RHS of the first node.
     * @param their_lhs The LHS of the second node.
     * @param their_rhs The RHS of the second node.
     * @return Does the first LHS equal the LHS of the second node, and does the first RHS equal the RHS of
     * the second node?
     */
    template<class LHSType, class RHSType>
        requires WeaklyEqualityComparableWith<LHSType, RHSType>
    [[nodiscard]] static bool noncommutative_ptr_compare(const LHSType *const my_lhs,
            const RHSType *const my_rhs, const LHSType *const their_lhs,
            const RHSType *const their_rhs) noexcept
    {
        bool lhs_matches = false;

        if (my_lhs == nullptr)
            // My LHS is NULL, so equality is achieved if and only if their LHS is also NULL.
            lhs_matches = their_lhs == nullptr;
        else if (their_lhs == nullptr)
            // Their LHS is NULL, but we know that our LHS is non-NULL.
            lhs_matches = false;
        else
            // Both LHS pointers are non-NULL.
            lhs_matches = *my_lhs == *their_lhs;

        if (!lhs_matches)
            return false;

        // LHS matches. If either of the RHS pointers are NULL, equality is achieved if and only if they're
        // both NULL.
        if (my_rhs == nullptr || their_rhs == nullptr)
            return my_rhs == their_rhs;

        // Both RHS pointers are non-NULL.
        return *my_rhs == *their_rhs;
    }
};

} // namespace optifol

#endif
```

1.11.4.7 Literal.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Literal IR node
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#include "Literal.hpp"

#include <algorithm>
#include <ranges>

#include "../../CompositeSerialisationHelpers.hpp"
#include "../../Visitors/RegularTargets/FeatureBuildingVisitor.hpp"
#include "../../Visitors/RegularTargets/IObservingBinaryVisitor.hpp"

namespace optifol
{

Literal::Literal(std::string name, const std::initializer_list<const IProcessedTerm *> arguments,
        const bool is_positive) :
    name(std::move(name)),
    arguments(arguments),
    is_positive(is_positive)
```

```
{
}

Literal::Literal(std::string name, std::vector<const IProcessedTerm *> &&arguments, const bool is_positive) :
    name(std::move(name)),
    arguments(std::move(arguments)),
    is_positive(is_positive)
{
}

bool Literal::is_negative_polarity() const noexcept
{
    return !is_positive;
}

std::ostream &Literal::serialise(std::ostream &ostream) const
{
    return CompositeSerialisationHelpers::stream_serialise(
            ostream, name, arguments.cbegin(), arguments.cend(), is_negative_polarity());
}

std::size_t Literal::hash() const noexcept
{
    return composite_hash(name, arguments.cbegin(), arguments.cend(), is_negative_polarity());
}

bool Literal::operator==(const IProcessedSentence &other) const noexcept
{
    return is_negative_polarity() == other.is_negative_polarity() && unsigned_equality(other);
}

bool Literal::operator<(const Literal &other) const noexcept
{
    // Order lexicographically on the string_view.
    if (get_name() < other.get_name())
        return true;

    if (get_name() > other.get_name())
        return false;

    // Names are lexicographically equal. Order on sign; negatives are considered less.
    if (is_negative_polarity() && !other.is_negative_polarity())
        return true;

    if (!is_negative_polarity() && other.is_negative_polarity())
        return false;

    // Literals are superficially equal, so order based on arguments.
    return std::ranges::any_of(std::ranges::views::zip(arguments, other.arguments),
            [](const auto &pair) { return *std::get<0>(pair) < *std::get<1>(pair); });
}

std::string_view Literal::get_name() const noexcept
{
    return name;
}

const std::vector<const IProcessedTerm *> &Literal::observe_arguments() const noexcept
{
    return arguments;
}

bool Literal::accept(IObservingBinaryVisitor &binary_visitor, const IProcessedSentence &sentence) const
{
    return sentence.accept(binary_visitor, *this);
}

bool Literal::accept(IObservingBinaryVisitor &binary_visitor, const Literal &predicate) const
{
    return binary_visitor.visit(*this, predicate);
}

void Literal::accept(FeatureBuildingVisitor &feature_component_builder) const
{
    feature_component_builder.visit(this);
}

const Literal *Literal::accept(const UnificationApplicationVisitor &application_visitor) const
{
    return application_visitor.visit(*this);
}
```

```cpp
bool Literal::unsigned_equality(const IProcessedSentence &other) const noexcept
{
    const auto other_literal = dynamic_cast<const Literal *>(&other);
    if (other_literal == nullptr)
        // Other IProcessedSentence isn't a Literal.
        return false;

    if (name != other_literal->name)
        // Other literal has a different superficial name.
        return false;

    const auto argument_count = arguments.size();
    if (argument_count != other_literal->arguments.size())
        // Other literal has a different number of arguments.
        return false;

    for (std::size_t arg_idx = 0; arg_idx < argument_count; ++arg_idx)
        if (*arguments[arg_idx] != *other_literal->arguments[arg_idx])
            // Other pairwise argument is different according to its own comparator.
            return false;

    return true;
}

std::ostream &operator<<(std::ostream &ostream, const Literal &literal)
{
    return literal.serialise(ostream);
}

} // namespace optifol
```

1.11.4.8 Literal.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Literal IR node
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#ifndef LITERAL_HPP
#define LITERAL_HPP

#include <vector>

#include "../../Visitors/RegularTargets/Unification/UnificationApplicationVisitor.hpp"
#include "../MutableVariants/OwningBuildable.hpp"
#include "IProcessedSentence.hpp"

namespace optifol
{

class IProcessedTerm;
class BidirectionalUnificationVisitor;
class UnificationApplicationVisitor;
class FeatureBuildingVisitor;

/**
 * @class Literal
 * @brief A Literal represents a non-owning IR node sentence consisting of a display name and zero to many
 * arguments, all of which are referenced from the centralised SymbolRepository.
 * @see MutablePredicate for the unprocessed, argument-owning dual.
 */
class Literal : public IProcessedSentence,
                public OwningBuildable<Literal>
{
public:
    /**
     * @brief Create a signed predicate with an initial set of referenced arguments
     * @param name Literal display name
     * @param arguments Set of non-owning pointers to immutable arguments
     * @param is_positive Should the predicate be instantiated with a positive polarity?
```

```
     */
    explicit Literal(std::string name, std::initializer_list<const IProcessedTerm *> arguments = {},
            bool is_positive = true);

    /**
     * @brief Create a signed predicate with an initial set of referenced arguments
     * @param name Literal display name
     * @param arguments Set of non-owning pointers to immutable arguments
     * @param is_positive Should the predicate be instantiated with a positive polarity?
     */
    explicit Literal(
            std::string name, std::vector<const IProcessedTerm *> &&arguments, bool is_positive = true);

    [[nodiscard]] bool is_negative_polarity() const noexcept override;

    std::ostream &serialise(std::ostream &ostream) const override;

    friend std::ostream &operator<<(std::ostream &ostream, const Literal &literal);

    [[nodiscard]] std::size_t hash() const noexcept override;

    [[nodiscard]] bool operator==(const IProcessedSentence &other) const noexcept override;

    [[nodiscard]] bool operator<(const Literal &other) const noexcept;

    /**
     * @brief Get the display name of the predicate, not including any arguments or metadata
     * @return A view of the predicate symbol name
     */
    [[nodiscard]] std::string_view get_name() const noexcept;

    /**
     * @brief Observe the non-owning ordered argument collection
     * @return The arguments referenced by the predicate
     */
    [[nodiscard]] const std::vector<const IProcessedTerm *> &observe_arguments() const noexcept;

    [[nodiscard]] bool accept(
            IObservingBinaryVisitor &binary_visitor, const IProcessedSentence &sentence) const override;

    [[nodiscard]] bool accept(
            IObservingBinaryVisitor &binary_visitor, const Literal &predicate) const override;

    void accept(FeatureBuildingVisitor &feature_component_builder) const;

    [[nodiscard]] const Literal *accept(const UnificationApplicationVisitor &application_visitor) const;

    [[nodiscard]] bool unsigned_equality(const IProcessedSentence &other) const noexcept;

private:
    const std::string name;
    const std::vector<const IProcessedTerm *> arguments;
    const bool is_positive = true;
};

} // namespace optifol

#endif
```

1.11.4.9 SentenceRoot.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the mutable Sentence Root IR node
 * @author Oliver Dixon
 * @date 2025-06-23
 * @version Development
 */

#include "SentenceRoot.hpp"

#include <algorithm>

#include "Literal.hpp"
```

```
namespace optifol
{

bool SentenceRoot::is_negative_polarity() const noexcept
{
    return false;
}

std::ostream &SentenceRoot::serialise(std::ostream &ostream) const
{
    ostream << '{' << ' ';

    if (!clauses.empty()) {
        std::ranges::for_each_n(clauses.begin(), clauses.size() - 1,
                [&ostream](const Clause &clause) { ostream << clause << ',' << ' '; });

        ostream << clauses.back();
    }

    return ostream << ' ' << '}';
}

std::size_t SentenceRoot::hash() const noexcept
{
    std::size_t hash_value = clauses.size();

    for (const auto &clause: clauses) {
        std::size_t literal_hash_value = clause.order();
        for (const auto literal: clause)
            literal_hash_value = hash_combine(literal_hash_value, literal->hash());

        hash_value = hash_combine_commutative(hash_value, literal_hash_value);
    }

    return hash_value;
}

bool SentenceRoot::operator==(const IProcessedSentence &other) const noexcept
{
    const auto other_root = dynamic_cast<const SentenceRoot *>(&other);
    if (other_root == nullptr)
        // Other IProcessedSentence isn't a SentenceRoot.
        return false;

    // We have an ordering on the Clauses, so this is OK.
    return std::ranges::equal(clauses, other_root->clauses);
}

void SentenceRoot::add_clause(Clause new_clause)
{
    /*
     * Don't use std::ranges::lower_bound here. Clause doesn't model std::totally_ordered_with because it
     * defines only operator<, hence std::ranges::less cannot be used, and it's not worth the hassle of
     * defining a custom comparator here.
     */
    const auto nearest_lower = std::lower_bound(clauses.begin(), clauses.end(), new_clause);

    if ((nearest_lower != clauses.end() && *nearest_lower == new_clause) ||
            (nearest_lower != clauses.begin() && *std::prev(nearest_lower) == new_clause))
        /*
         * Reject the new clause if it already exists within the SentenceRoot. Note that the only two
         * positions at which it could appear, assuming correct lexicographical ordering provided by
         * Clause::operator<, is at the proposed insertion point or immediately prior.
         */
        return;

    clauses.insert(nearest_lower, std::move(new_clause));
}

std::vector<Clause>::const_iterator SentenceRoot::begin() const noexcept
{
    return clauses.begin();
}

std::vector<Clause>::const_iterator SentenceRoot::end() const noexcept
{
    return clauses.end();
}

std::size_t SentenceRoot::order() const noexcept
```

```
{
    return clauses.size();
}

} // namespace optifol
```

1.11.4.10 SentenceRoot.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Sentence Root IR node
 * @author Oliver Dixon
 * @date 2025-06-23
 * @version Development
 */

#ifndef SENTENCEROOT_HPP
#define SENTENCEROOT_HPP

#include "Clause.hpp"
#include "IProcessedSentence.hpp"

namespace optifol
{

/**
 * @class SentenceRoot
 * @brief A SentenceRoot denotes the root node of an immutable IR node set. It contains conjunctive clauses of
 * literals under disjunction, where the literals are weak references to a lifetime-assured SymbolRepository.
 */
class SentenceRoot : public IProcessedSentence,
                     public OwningBuildable<SentenceRoot>
{
public:
    [[nodiscard]] bool is_negative_polarity() const noexcept override;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    [[nodiscard]] bool operator==(const IProcessedSentence &other) const noexcept override;

    void add_clause(Clause new_clause);

    [[nodiscard]] std::vector<Clause>::const_iterator begin() const noexcept;

    [[nodiscard]] std::vector<Clause>::const_iterator end() const noexcept;

    [[nodiscard]] std::size_t order() const noexcept;

private:
    std::vector<Clause> clauses;

    bool is_negative = false;
};

} // namespace optifol

#endif // SENTENCEROOT_HPP
```

1.11.5 Terms

1.11.5.1 Function.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Function IR node
```

```
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#include "Function.hpp"

#include <algorithm>
#include <ranges>

#include "../../CompositeSerialisationHelpers.hpp"
#include "../../Visitors/RegularTargets/FeatureBuildingVisitor.hpp"
#include "../../Visitors/RegularTargets/Unification/BidirectionalUnificationVisitor.hpp"

namespace optifol
{

Function::Function(std::string name, const std::initializer_list<const IProcessedTerm *> arguments) :
    name(std::move(name)),
    arguments(arguments)
{
}

Function::Function(std::string name, std::vector<const IProcessedTerm *> &&arguments) :
    name(std::move(name)),
    arguments(std::move(arguments))
{
}

const std::vector<const IProcessedTerm *> &Function::observe_arguments() const noexcept
{
    return arguments;
}

bool Function::accept(IObservingBinaryVisitor &binary_visitor, const IProcessedTerm &term) const
{
    return term.accept_reverse(binary_visitor, *this);
}

bool Function::accept(IObservingBinaryVisitor &binary_visitor, const Function &function) const
{
    return binary_visitor.visit(*this, function);
}

bool Function::accept(IObservingBinaryVisitor &binary_visitor, const Variable &variable) const
{
    return binary_visitor.visit(*this, variable);
}

bool Function::accept_reverse(IObservingBinaryVisitor &binary_visitor, const Function &function) const
{
    return binary_visitor.visit(function, *this);
}

bool Function::accept_reverse(IObservingBinaryVisitor &binary_visitor, const Variable &variable) const
{
    return binary_visitor.visit(variable, *this);
}

bool Function::is_self_nested(const IProcessedTerm &search_term) const noexcept
{
    return std::ranges::any_of(arguments, [&search_term](const IProcessedTerm *const argument)
            { return argument->is_self_nested(search_term); });
}

void Function::accept(FeatureBuildingVisitor &feature_building_visitor) const noexcept
{
    feature_building_visitor.visit(this);
}

const IProcessedTerm *Function::accept(
        const UnificationApplicationVisitor &unification_application_visitor) const
{
    return unification_application_visitor.visit(*this);
}

bool Function::operator==(const IProcessedTerm &other) const noexcept
{
    const auto other_function = dynamic_cast<const Function *>(&other);
    if (other_function == nullptr)
        // Other IProcessedTerm isn't a MutableFunction.
```

```
        return false;

    if (get_disambiguated_name() != other_function->get_disambiguated_name())
        // Different superficial names.
        return false;

    const auto argument_count = arguments.size();
    if (argument_count != other_function->arguments.size())
        // Different number of arguments.
        return false;

    for (std::size_t argument_idx = 0; argument_idx < argument_count; ++argument_idx) {
        const auto &lhs_arg_ptr = arguments[argument_idx];
        const auto &rhs_arg_ptr = other_function->arguments[argument_idx];

        if (lhs_arg_ptr == nullptr) {
            if (rhs_arg_ptr != nullptr)
                return false;
        } else if (rhs_arg_ptr == nullptr)
            return false;
        else if (*lhs_arg_ptr != *rhs_arg_ptr)
            return false;
    }

    return true;
}
bool Function::operator<(const IProcessedTerm &other) const noexcept
{
    const auto other_function = dynamic_cast<const Function *>(&other);
    if (other_function == nullptr)
        // Other IProcessedTerm isn't a MutableFunction.
        return false;

    return std::ranges::any_of(std::views::zip(arguments, other_function->arguments),
            [](const auto &pair) { return *std::get<0>(pair) < *std::get<1>(pair); });
}

std::string Function::to_string() const
{
    return CompositeSerialisationHelpers::string_serialise(name, arguments.cbegin(), arguments.cend());
}

std::string_view Function::get_disambiguated_name() const
{
    return name;
}

std::ostream &Function::serialise(std::ostream &ostream) const
{
    return CompositeSerialisationHelpers::stream_serialise(
            ostream, name, arguments.cbegin(), arguments.cend());
}

std::size_t Function::hash() const noexcept
{
    return composite_hash(name, arguments.cbegin(), arguments.cend());
}

} // namespace optifol
```

1.11.5.2 Function.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Function IR node
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#ifndef FUNCTION_HPP
#define FUNCTION_HPP

#include <vector>
```

```
#include "../MutableVariants/OwningBuildable.hpp"
#include "IProcessedTerm.hpp"

namespace optifol
{

/**
 * @class Function
 * @brief A Function represents a non-owning IR node term consisting of a display name and zero to many
 * arguments, all of which are referenced from the centralised SymbolRepository.
 * @see MutableFunction for the unprocessed, argument-owning dual.
 */
class Function : public IProcessedTerm,
                 public OwningBuildable<Function>
{
public:
    /**
     * @brief Create a non-owning unsigned function with an initial set of referenced immutable arguments.
     * @param name Display name of the function
     * @param arguments Set of non-owning pointers to immutable arguments
     */
    [[maybe_unused]] explicit Function(
            std::string name, std::initializer_list<const IProcessedTerm *> arguments = {});

    explicit Function(std::string name, std::vector<const IProcessedTerm *> &&arguments);

    [[nodiscard]] std::string to_string() const override;

    [[nodiscard]] std::string_view get_disambiguated_name() const override;

    std::ostream &serialise(std::ostream &ostream) const override;

    [[nodiscard]] std::size_t hash() const noexcept override;

    /**
     * @brief Observe the non-owning ordered argument collection
     * @return The arguments referenced by the function
     */
    [[nodiscard]] const std::vector<const IProcessedTerm *> &observe_arguments() const noexcept;

    [[nodiscard]] bool accept(
            IObservingBinaryVisitor &binary_visitor, const IProcessedTerm &term) const override;

    [[nodiscard]] bool accept(
            IObservingBinaryVisitor &binary_visitor, const Function &function) const override;
    [[nodiscard]] bool accept(
            IObservingBinaryVisitor &binary_visitor, const Variable &variable) const override;

    [[nodiscard]] bool accept_reverse(
            IObservingBinaryVisitor &binary_visitor, const Function &function) const override;
    [[nodiscard]] bool accept_reverse(
            IObservingBinaryVisitor &binary_visitor, const Variable &variable) const override;

    [[nodiscard]] bool is_self_nested(const IProcessedTerm &search_term) const noexcept override;

    void accept(FeatureBuildingVisitor &feature_building_visitor) const noexcept override;
    [[nodiscard]] const IProcessedTerm *accept(
            const UnificationApplicationVisitor &unification_application_visitor) const override;

    [[nodiscard]] bool operator==(const IProcessedTerm &other) const noexcept override;

    [[nodiscard]] bool operator<(const IProcessedTerm &other) const noexcept override;

private:
    const std::string name;
    const std::vector<const IProcessedTerm *> arguments;
};

} // namespace optifol

#endif
```

1.11.5.3 IProcessedTerm.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */
```

```cpp
/**
 * @file
 * @brief Class specification for the IR Processed Term interface
 * @author Oliver Dixon
 * @date 2025-06-14
 * @version Development
 */

#ifndef IPROCESSEDTERM_HPP
#define IPROCESSEDTERM_HPP

#include "ITerm.hpp"

namespace optifol
{

class FeatureBuildingVisitor;

class IObservingBinaryVisitor;
class UnificationApplicationVisitor;
class RepositoryBuildingVisitor;

class Function;
class Variable;

/**
 * @class IProcessedTerm
 * @brief An IProcessedTerm is an ITerm that has undergone the lexing, parsing, and normalisation pipeline and
 * is now held, in its disambiguated form, in a centralised SymbolRepository.
 * @details Such terms are always immutable and have lifetime guarantees in accordance with their responsible
 *  SymbolRepository.
 */
class IProcessedTerm : public ITerm
{
public:
    /**
     * @brief Accept a visitation from an IObservingBinaryVisitor, despatching dynamically on the given term
     * as the second operand.
     * @param binary_visitor The binary visitor from whom to accept a visit.
     * @param term The generic term for the second operand.
     * @return Visitor return code.
     */
    [[nodiscard]] virtual bool accept(
            IObservingBinaryVisitor &binary_visitor, const IProcessedTerm &term) const = 0;

    /**
     * @brief Accept a visitation from an IObservingBinaryVisitor, using the given Function as the second
     * operand.
     * @param binary_visitor The binary visitor from whom to accept a visit.
     * @param function The Function term for the second operand.
     * @return Visitor return code.
     */
    [[nodiscard]] virtual bool accept(
            IObservingBinaryVisitor &binary_visitor, const Function &function) const = 0;

    /**
     * @brief Accept a visitation from an IObservingBinaryVisitor, using the given Variable as the second
     * operand.
     * @param binary_visitor The binary visitor from whom to accept a visit.
     * @param variable The Variable term for the second operand.
     * @return Visitor return code.
     */
    [[nodiscard]] virtual bool accept(
            IObservingBinaryVisitor &binary_visitor, const Variable &variable) const = 0;

    virtual void accept(FeatureBuildingVisitor &feature_building_visitor) const noexcept = 0;

    /**
     * @brief Accept a visitation from an IObservingBinaryVisitor, using the given Function as the first
     * operand.
     * @param binary_visitor The binary visitor from whom to accept a visit.
     * @param function The Function term for the first operand.
     * @return Visitor return code.
     */
    [[nodiscard]] virtual bool accept_reverse(
            IObservingBinaryVisitor &binary_visitor, const Function &function) const = 0;

    /**
     * @brief Accept a visitation from an IObservingBinaryVisitor, using the given Variable as the first
     * operand.
```

```
     * @param binary_visitor The binary visitor from whom to accept a visit.
     * @param variable The Variable term for the first operand.
     * @return Visitor return code.
     */
    [[nodiscard]] virtual bool accept_reverse(
            IObservingBinaryVisitor &binary_visitor, const Variable &variable) const = 0;

    /**
     * @brief Determines whether the given Variable appears in the expansion of the term.
     * @param search_term The Variable for which to search in the term.
     * @return Does the given Variable appear in the IProcessedTerm or any trivial expansion thereof?
     */
    [[nodiscard]] virtual bool is_self_nested(const IProcessedTerm &search_term) const noexcept = 0;

    /**
     * @brief Accept a visit from the UnificationApplicationVisitor to construct new IProcessedTerm objects by
     *  substituting Variable instances into the term.
     * @param unification_application_visitor The non-mutating visitor to accept.
     * @see UnificationApplicationVisitor for the visitor context.
     */
    [[nodiscard]] virtual const IProcessedTerm *accept(
            const UnificationApplicationVisitor &unification_application_visitor) const = 0;

    /**
     * @brief Test equality between two IProcessedTerm instances.
     * @param other The other IProcessedTerm instance with which to test for equality.
     * @return Are the two terms of the same concrete type and equal?
     */
    [[nodiscard]] virtual bool operator==(const IProcessedTerm &other) const noexcept = 0;

    /**
     * @brief Test ordering between two IProcessedTerm instances.
     * @param other The other IProcessedTerm instance to compare against.
     * @return Am I strictly less than the other term?
     */
    [[nodiscard]] virtual bool operator<(const IProcessedTerm &other) const noexcept = 0;
};

} // namespace optifol

#endif // IPROCESSEDTERM_HPP
```

1.11.5.4 ITerm.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the generic IR Term interface
 * @author Oliver Dixon
 * @date 2025-06-14
 * @version Development
 */

#ifndef ITERM_HPP
#define ITERM_HPP

#include "../../IHashable.hpp"
#include "../../ISerialisable.hpp"

namespace optifol
{

class Variable;
class Function;

/**
 * @class ITerm
 * @brief The ITerm interface describes a generic term in first-order logic.
 * @details
 *  <p>
 *      Typical FOL terms include:
 *      <ul>
 *          <li>Functions (inc. introduced Skolem functions)</li>
 *          <li>Bound variables</li>
 *      </ul>
```

```
 *  </p>
 *  <p>
 *      All terms contain a display name, which is guaranteed to match the symbol identifier provided by the
 * user. Terms also include a disambiguated name, which may or may not be equal to the display name and is
 * resolved internally to guarantee uniqueness within the scope of a single sentence. More precisely, the
 * disambiguated name is unique up to the outermost IR node, but not necessarily across multiple SentenceRoot
 * or ITerm root objects.
 *  </p>
 */
class ITerm : public IHashable,
              public ISerialisable
{
public:
    [[nodiscard]] std::size_t hash() const noexcept override
    {
        return std::hash<std::string>{}(to_string());
    }

    /**
     * @brief Create a human-readable @ref std::string representing the term, including all child terms.
     * @return The constructed string representation of the term
     */
    [[nodiscard]] virtual std::string to_string() const = 0;

    /**
     * @brief Create a view of the disambiguated term identifier
     * @return The observing string view of the disambiguated name
     */
    [[nodiscard]] virtual std::string_view get_disambiguated_name() const = 0;

    /**
     * @brief Serialise the ITerm to a destination output @ref std::ostream stream
     * @param ostream The destination output stream
     * @return The populated destination output stream
     */
    std::ostream &serialise(std::ostream &ostream) const override
    {
        return ostream << to_string();
    }

    /**
     * @brief Test hash-based equality with another ITerm
     * @param other The ITerm with which equivalence should be tested
     * @return Are the ITerm objects equal?
     */
    bool operator==(const ITerm &other) const
    {
        return hash() == other.hash();
    }
};

} // namespace optifol

#endif
```

1.11.5.5 SkolemFunction.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Skolem Function IR node
 * @author Oliver Dixon
 * @date 2025-01-30
 * @version Development
 */

#include "SkolemFunction.hpp"

#include "../../Visitors/RegularTargets/FeatureBuildingVisitor.hpp"

namespace optifol
{

SkolemFunction::SkolemFunction(std::string name, std::initializer_list<const IProcessedTerm *> arguments) :
    Function(std::move(name), arguments)
```

```
{
}

SkolemFunction::SkolemFunction(std::string name, std::vector<const IProcessedTerm *> &&arguments) :
    Function(std::move(name), std::move(arguments))
{
}

void SkolemFunction::accept(FeatureBuildingVisitor &feature_building_visitor) const noexcept
{
    feature_building_visitor.visit(this);
}

} // namespace optifol
```

1.11.5.6 SkolemFunction.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Skolem Function IR node
 * @author Oliver Dixon
 * @date 2025-01-30
 * @version Development
 */

#ifndef OPTIFOL_SKOLEMFUNCTION_HPP
#define OPTIFOL_SKOLEMFUNCTION_HPP

#include "Function.hpp"

namespace optifol
{

/**
 * @class SkolemFunction
 * @brief A Skolem Function is a Function symbol used to preserve equivalence with existentially quantified
 * expressions in CNF. It is typically introduced by the normalisation pipeline.
 */
class SkolemFunction : public Function
{
public:
    [[maybe_unused]] explicit SkolemFunction(
            std::string name, std::initializer_list<const IProcessedTerm *> arguments = {});

    explicit SkolemFunction(std::string name, std::vector<const IProcessedTerm *> &&arguments);

    void accept(FeatureBuildingVisitor &feature_building_visitor) const noexcept override;
};

} // namespace optifol

#endif // OPTIFOL_SKOLEMFUNCTION_HPP
```

1.11.5.7 Variable.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Function IR node
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#include "Variable.hpp"

#include "../../Visitors/RegularTargets/FeatureBuildingVisitor.hpp"
#include "../../Visitors/RegularTargets/Unification/BidirectionalUnificationVisitor.hpp"
```

```
namespace optifol
{

Variable::Variable(std::string name) :
    name(std::move(name))
{
}

Variable::Variable(std::string name, std::string disambiguated_name) :
    name(std::move(name)),
    disambiguated_name(std::move(disambiguated_name))
{
}

std::string Variable::to_string() const
{
    return std::string(get_disambiguated_name());
}

std::string_view Variable::get_disambiguated_name() const
{
    if (disambiguated_name.has_value())
        return *disambiguated_name;

    return name;
}

std::string_view Variable::get_base_name() const noexcept
{
    return name;
}

bool Variable::operator==(const Variable &other) const
{
    return disambiguated_name == other.disambiguated_name;
}

bool Variable::accept(IObservingBinaryVisitor &binary_visitor, const IProcessedTerm &term) const
{
    return term.accept_reverse(binary_visitor, *this);
}

bool Variable::accept(IObservingBinaryVisitor &binary_visitor, const Function &function) const
{
    return binary_visitor.visit(*this, function);
}

bool Variable::accept(IObservingBinaryVisitor &binary_visitor, const Variable &variable) const
{
    return binary_visitor.visit(*this, variable);
}

bool Variable::accept_reverse(IObservingBinaryVisitor &binary_visitor, const Function &function) const
{
    return binary_visitor.visit(function, *this);
}

bool Variable::accept_reverse(IObservingBinaryVisitor &binary_visitor, const Variable &variable) const
{
    return binary_visitor.visit(variable, *this);
}

bool Variable::is_self_nested(const IProcessedTerm &search_term) const noexcept
{
    return search_term == *this;
}

void Variable::accept(FeatureBuildingVisitor &feature_building_visitor) const noexcept
{
    feature_building_visitor.visit(this);
}

const IProcessedTerm *Variable::accept(
        const UnificationApplicationVisitor &unification_application_visitor) const
{
    return unification_application_visitor.visit(*this);
}

bool Variable::operator==(const IProcessedTerm &other) const noexcept
{
```

```
    const auto other_variable = dynamic_cast<const Variable *>(&other);
    if (other_variable == nullptr)
        // Other IProcessedTerm isn't a Variable.
        return false;

    return get_disambiguated_name() == other_variable->get_disambiguated_name();
}

bool Variable::operator<(const IProcessedTerm &other) const noexcept
{
    const auto other_variable = dynamic_cast<const Variable *>(&other);
    if (other_variable == nullptr)
        // Other IProcessedTerm isn't a Variable.
        return false;

    return get_disambiguated_name() < other.get_disambiguated_name();
}

} // namespace optifol
```

1.11.5.8 Variable.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Variable IR node
 * @author Oliver Dixon
 * @date 2025-06-15
 * @version Development
 */

#ifndef VARIABLE_HPP
#define VARIABLE_HPP

#include <optional>

#include "../MutableVariants/OwningBuildable.hpp"
#include "IProcessedTerm.hpp"

namespace optifol
{

/**
 * @class Variable
 * @brief A Variable is a non-owning IR node representing a first-order logic non-free variable.
 * @see MutableVariable for the owning, mutable dual; MutableVariable also contains more documentation of the
 * semantics of an Optifol first-order logic variable.
 */
class Variable : public IProcessedTerm,
                 public OwningBuildable<Variable>
{
public:
    /**
     * @brief Create a new Variable with a fixed display name
     * @param name The fixed name of the variable
     */
    explicit Variable(std::string name);

    /**
     * @brief Create a new Variable with a fixed display name and disambiguated name
     * @param name The fixed name of the variable
     * @param disambiguated_name The fixed disambiguated name for the variable
     * @warning No uniqueness check is done for the disambiguated name upon construction
     */
    explicit Variable(std::string name, std::string disambiguated_name);

    [[nodiscard]] std::string to_string() const override;

    [[nodiscard]] std::string_view get_disambiguated_name() const override;

    [[nodiscard]] std::string_view get_base_name() const noexcept;

    [[nodiscard]] bool operator==(const Variable &other) const;

    [[nodiscard]] bool accept(
```

```
            IObservingBinaryVisitor &binary_visitor, const IProcessedTerm &term) const override;

    [[nodiscard]] bool accept(
            IObservingBinaryVisitor &binary_visitor, const Function &function) const override;
    [[nodiscard]] bool accept(
            IObservingBinaryVisitor &binary_visitor, const Variable &variable) const override;

    [[nodiscard]] bool accept_reverse(
            IObservingBinaryVisitor &binary_visitor, const Function &function) const override;
    [[nodiscard]] bool accept_reverse(
            IObservingBinaryVisitor &binary_visitor, const Variable &variable) const override;

    /**
     * @copydoc IProcessedTerm::is_self_nested
     * @return Always false, as Variable objects are not nestable.
     */
    [[nodiscard]] bool is_self_nested(const IProcessedTerm &search_term) const noexcept override;

    void accept(FeatureBuildingVisitor &feature_building_visitor) const noexcept override;
    [[nodiscard]] const IProcessedTerm *accept(
            const UnificationApplicationVisitor &unification_application_visitor) const override;

    [[nodiscard]] bool operator==(const IProcessedTerm &other) const noexcept override;

    [[nodiscard]] bool operator<(const IProcessedTerm &other) const noexcept override;

private:
    const std::string name;
    const std::optional<std::string> disambiguated_name;
};

} // namespace optifol

#endif
```

1.12 Reporting

1.12.1 IReportGenerator.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the report generator interface
 * @author Oliver Dixon
 * @date 2025-08-05
 * @version Development
 */

#ifndef OPTIFOL_IREPORTGENERATOR_HPP
#define OPTIFOL_IREPORTGENERATOR_HPP

namespace optifol
{

class Requirement;
class TestGroup;

/**
 * @class IReportGenerator
 * @brief The report generator interface provides a set of operations for generating reports based on
 * subsystems, enumerating their constituent Requirements index, analysis results, and automated test
 * evidence.
 */
class IReportGenerator
{
public:
    /**
     * @brief Destruct the report generator instance, flushing and releasing any open system resources.
     */
    virtual ~IReportGenerator() = default;

    /**
     * @brief Add a Requirement to the requirements index.
     * @param requirement The Requirement to serialise into the report index.
     */
```

```
    virtual void add_requirement(const Requirement &requirement) = 0;

    /**
     * @brief Add a TestGroup, and all containing Requirement and Test objects, to the automated tests
     * section.
     * @param test_group The TestGroup to serialise.
     */
    virtual void add_test_group(const TestGroup &test_group) = 0;

    /**
     * @brief Asynchronously finalise the generation of the report, streaming output to the given buffer.
     * @param output The Gtk::TextBuffer to which compilation output should be streamed.
     * @param finished_callback The callback to execute upon completion of the asynchronous generation.
     */
    virtual void generate(Glib::RefPtr<Gtk::TextBuffer> output, sigc::slot<void()> &&finished_callback) = 0;
};

} // namespace optifol

#endif // OPTIFOL_IREPORTGENERATOR_HPP
```

1.12.2 LaTeXReportGenerator.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the LaTeX report-generator backend
 * @author Oliver Dixon
 * @date 2025-08-05
 * @version Development
 */

#include <cassert>
#include <giomm/file.h>
#include <glibmm/miscutils.h>

#include "../Storage/Requirement.hpp"
#include "../UserTesting/Modelling/TestGroup.hpp"
#include "LaTeXReportGenerator.hpp"

namespace optifol
{

LaTeXReportGenerator::LaTeXReportGenerator(const Glib::RefPtr<Gio::File> &output_directory) :
    output_directory(output_directory),
    index_file(Gio::File::create_for_path(this->output_directory->get_path() + "/index.tex")->append_to()),
    tests_file(Gio::File::create_for_path(this->output_directory->get_path() + "/tests.tex")->append_to())
{
    start_requirements();
}

void LaTeXReportGenerator::add_requirement(const Requirement &requirement)
{
    write_property(*index_file, requirement.property_name());
    write_property(*index_file, requirement.property_description());
    write_property(*index_file, std::string(requirement.observe_latex_statement()), true);
    write_property(*index_file, serialise_time(requirement.property_creation_time().get_value()), true);
    write_property(*index_file, serialise_time(requirement.property_modified_time().get_value()), true, true);
}

void LaTeXReportGenerator::add_test_group(const TestGroup &test_group)
{
    tests_file->write("\n\\subsection{" + test_group.property_name().get_value() + '}');
    test_group.for_each(
            [this](const Requirement &requirement)
            {
                tests_file->write("\\subsubsection{" + requirement.property_name().get_value() + "}");
                tests_file->write(R"(
\begin{xltabular}{\linewidth}{lXXl}%
                \toprule\normalfont%
            \textbf{Test Name}&%
            \textbf{Target Executable}&%
            \textbf{Test Fixture}&%
            \textbf{Test Result}%
        \\\midrule\endhead%
```

```
                \bottomrule\endfoot%
)");

                const auto tests = requirement.observe_tests();
                const auto test_count = tests->get_n_items();

                for (guint test_index = 0; test_index < test_count; ++test_index) {
                    const auto test = tests->get_item(test_index);
                    const auto result = test->property_result().get_value();

                    write_property(*tests_file, test->property_name());
                    write_property(*tests_file, test->property_target_executable_name());
                    write_property(*tests_file, test->property_fixture().get_value());

                    if (result == nullptr)
                        write_property(*tests_file, "", true, true);
                    else
                        write_property(*tests_file, result->has_passed() ? "Passed" : "Failed", true, true);
                }

                tests_file->write("\\end{xltabular}%\n");
            });
}

void LaTeXReportGenerator::generate(
        Glib::RefPtr<Gtk::TextBuffer> output, sigc::slot<void()> &&finished_callback)
{
    assert(index_file->is_closed() == false);
    assert(tests_file->is_closed() == false);

    // Close the files.
    end_requirements();
    tests_file->close();

    // Start the sub-process.
    latex_executor.emplace("",
            std::vector<std::string>{"latexmk", "-pdf", "-interaction=nonstopmode",
                    "-outdir=" + output_directory->get_path(), "Resources/ReportTemplates/LaTeX/report.tex"},
            std::vector{"PATH=" + Glib::getenv("PATH"), "TEXINPUTS=.:" + output_directory->get_path() + ":"},
            std::move(output),
            [user_callback = std::move(finished_callback)](const int exit_code)
            {
                /*
                 * The interface doesn't care about implementation-specific details such as existence of a
                 * sub-process or its system exit code. Throw it away. The sub-process executor will report
                 * any faults.
                 */
                std::ignore = exit_code;
                user_callback();
            });
}

void LaTeXReportGenerator::start_requirements() const
{
    assert(index_file->is_closed() == false);

    index_file->write(R"(\begin{xltabular}{\linewidth}{lXXll}%
                \toprule\normalfont%
            \textbf{Name}&%
            \textbf{Description}&%
            \textbf{Statement}&%
            \textbf{Created}&%
            \textbf{Modified}%
        \\\midrule\endhead%
                \bottomrule\endfoot%
)");
}

void LaTeXReportGenerator::end_requirements() const
{
    assert(index_file->is_closed() == false);

    index_file->write("\\end{xltabular}%\n");
    index_file->close();

    assert(index_file->is_closed() == true);
}

void LaTeXReportGenerator::write_property(Gio::FileOutputStream &output_stream, const std::string &string,
        const bool verbatim, const bool eol) noexcept
{
```

```
    assert(output_stream.is_closed() == false);

    if (string.empty())
        output_stream.write("\\emph{Not provided.}");
    else
        output_stream.write(verbatim ? string : "\\optifolescaped|" + string + '|');

    output_stream.write(eol ? "\\\\\\n" : "&");
}

std::string LaTeXReportGenerator::serialise_time(const StorageObjectBase::TimeT &time)
{
    const auto seconds_since_epoch = floor<std::chrono::seconds>(time.time_since_epoch());

    return std::format("\\DTMdisplay{{{:%Y}}}{{{:%m}}}{{{:%d}}}{{-1}}{{{:%H}}}{{{:%M}}}{{{:%S}}}{{1}}{{0}}",
            time, time, time, time, time,
            std::chrono::duration_cast<std::chrono::seconds>(seconds_since_epoch) % std::chrono::minutes(1));
}

} // namespace optifol
```

1.12.3 LaTeXReportGenerator.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the LaTeX report-generator backend
 * @author Oliver Dixon
 * @date 2025-08-05
 * @version Development
 */

#ifndef OPTIFOL_LATEXREPORTGENERATOR_HPP
#define OPTIFOL_LATEXREPORTGENERATOR_HPP

#include <giomm/fileoutputstream.h>

#include "../GUI/StreamingProcessExecutor.hpp"
#include "../Storage/StorageObjectBase.hpp"
#include "IReportGenerator.hpp"

template<typename Candidate>
concept SerialisableProperty = requires(const Glib::PropertyProxy_ReadOnly<Candidate> candidate) {
    { candidate.get_value() } -> std::convertible_to<Glib::ustring>;
};

namespace optifol
{

/**
 * @class LaTeXReportGenerator
 * @brief Provide a LaTeX backend for the IReportGenerator. The implementation takes responsibility for the
 * construction and compilation of the LaTeX document into a PDF using the standard TeX Live
 * <code>latexml</code> utility in a sub-process.
 * @see StreamingProcessExecutor for the <code>latexmk</code> asynchronous invocation mechanism.
 */
class LaTeXReportGenerator : public IReportGenerator
{
public:
    /**
     * @brief Construct a new LaTeXReportGenerator at the given root.
     * @param output_directory The root directory for LaTeX generated and auxiliary files.
     */
    explicit LaTeXReportGenerator(const Glib::RefPtr<Gio::File> &output_directory);

    /**
     * @brief Disable the copy-constructor as objects have locks on external system resources.
     */
    LaTeXReportGenerator(const LaTeXReportGenerator &) = delete;

    void add_requirement(const Requirement &requirement) override;

    void add_test_group(const TestGroup &test_group) override;

    /**
```

```
     * @copydoc IReportGenerator::generate
     * @details Asynchronously invokes a <code>latexmk</code> sub-process to compile the TeX-serialised IR
     * nodes into a complete PDF. The results from <code>stdout</code> and <code>stderr</code> of the
     * <code>latexmk</code> process are streamed into the given Gtk::TextBuffer in near-real-time. Upon
     * completion of the sub-process, the given user callback is invoked.
     * @pre The requirements index LaTeX file is open for writing.
     * @pre The tests report LaTeX file is open for writing.
     */
    void generate(Glib::RefPtr<Gtk::TextBuffer> output, sigc::slot<void()> &&finished_callback) override;

private:
    /**
     * @brief Serialise the TeX starting sequence for the requirements index.
     * @pre The requirements stream is open for writing.
     */
    void start_requirements() const;

    /**
     * @brief Serialise the TeX ending sequence for the requirements index.
     * @pre The requirements stream is open for writing.
     */
    void end_requirements() const;

    /**
     * @brief Serialise a single Glib property to the given LaTeX output stream.
     * @tparam PropertyType The property type
     * @param output_stream The destination output stream, open for writing.
     * @param property The string to serialise.
     * @param verbatim Should the string be passed in verbatim to the TeX interpreter, or wrapped in a
     * escaping environment?
     * @param eol Should an end-of-line (EOL) marker be appended to the field?
     */
    template<SerialisableProperty PropertyType>
    static void write_property(Gio::FileOutputStream &output_stream,
            const Glib::PropertyProxy_ReadOnly<PropertyType> property, const bool verbatim = false,
            const bool eol = false)
    {
        write_property(output_stream, property.get_value(), verbatim, eol);
    }

    /**
     * @brief Serialise a string to the given LaTeX output stream.
     * @param output_stream The destination output stream, open for writing.
     * @param string The string to serialise.
     * @param verbatim Should the string be passed in verbatim to the TeX interpreter, or wrapped in a
     * escaping environment?
     * @param eol Should an end-of-line (EOL) marker be appended to the field?
     * @pre The output stream is open for writing.
     */
    static void write_property(Gio::FileOutputStream &output_stream, const std::string &string,
            bool verbatim = false, bool eol = false) noexcept;

    /**
     * @brief Serialises a time point to the LaTeX <code>datetime2</code> format.
     * @details Syntax: <code>\\DTMdisplay{YYYY}{MM}{DD}{-1}{dd}{mm}{ss}{1}{0}</code>
     * @param time The time-point to serialise.
     * @return The LaTeX-serialised <code>datetime2</code> macro.
     */
    static std::string serialise_time(const StorageObjectBase::TimeT &time);

    const Glib::RefPtr<Gio::File> output_directory;

    const Glib::RefPtr<Gio::FileOutputStream> index_file;
    const Glib::RefPtr<Gio::FileOutputStream> tests_file;

    std::optional<StreamingProcessExecutor> latex_executor;
};

} // namespace optifol

#endif // OPTIFOL_LATEXREPORTGENERATOR_HPP
```

1.13 Storage

1.13.1 AnalysisGroup.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
```

```
 */

/**
 * @file
 * @brief Class implementation for an analysis grouping of requirements
 * @author Oliver Dixon
 * @date 2025-04-27
 * @version Development
 */

#include "AnalysisGroup.hpp"

#include <assert.h>

namespace optifol
{

AnalysisGroup::AnalysisGroup(const Glib::ustring &name, std::shared_ptr<SymbolRepository> symbol_repository) :
    Glib::ObjectBase("AnalysisGroup"),
    kb(std::move(symbol_repository)),
    ObjectGroup(sigc::mem_fun(*this, &AnalysisGroup::handle_group_model_change))
{
    property_name().set_value(name);
}

AnalysisGroup::AnalysisGroup(const Glib::ustring &name, BaseObjectType *cobject,
        const Glib::RefPtr<Gtk::Builder> &builder, std::shared_ptr<SymbolRepository> symbol_repository) :
    Glib::ObjectBase("AnalysisGroup"),
    StorageObjectBase(cobject, builder),
    ObjectGroup(sigc::mem_fun(*this, &AnalysisGroup::handle_group_model_change)),
    kb(std::move(symbol_repository))
{
    property_name().set_value(name);
}

Prover &AnalysisGroup::observe_prover_instance() noexcept
{
    return kb;
}

void AnalysisGroup::handle_group_model_change(
        const guint initial_index, const guint, const guint added_count)
{
    for (guint added_list_i = initial_index; added_list_i < initial_index + added_count; ++added_list_i) {
        const auto requirement = get_object_by_index(added_list_i);
        kb.tell(requirement->observe_prepared_sentence());
    }
}

} // namespace optifol
```

1.13.2 AnalysisGroup.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for an analysis grouping of requirements
 * @author Oliver Dixon
 * @date 2025-04-27
 * @version Development
 */

#ifndef ANALYSISGROUP_HPP
#define ANALYSISGROUP_HPP

#include "../Inference/Prover.hpp"
#include "ObjectGroup.hpp"
#include "Requirement.hpp"

namespace optifol
{

class AnalysisGroup : public StorageObjectBase,
                      public ObjectGroup<Requirement>
{
```

```
public:
    /**
     * @brief Create a new Analysis Group with the given name and register in the Glib GType system
     * @param name The initial name of the Analysis Group
     * @param symbol_repository The associated SymbolRepository for Clause objects used by Requirement nodes
     * in the AnalysisGroup.
     */
    explicit AnalysisGroup(const Glib::ustring &name, std::shared_ptr<SymbolRepository> symbol_repository);

    AnalysisGroup(const Glib::ustring &name, BaseObjectType *cobject,
            const Glib::RefPtr<Gtk::Builder> &builder, std::shared_ptr<SymbolRepository> symbol_repository);

    Prover &observe_prover_instance() noexcept;

private:
    void handle_group_model_change(guint initial_index, guint removed_count, guint added_count);

    Prover kb;
};

} // namespace optifol

#endif
```

1.13.3 ObjectGroup.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the dual object grouping model
 * @author Oliver Dixon
 * @date 2025-07-26
 * @version Development
 */

#ifndef OBJECTGROUP_HPP
#define OBJECTGROUP_HPP

#include <giomm/liststore.h>
#include <gtkmm/dropdown.h>
#include <gtkmm/singleselection.h>

#include "../Optifol.hpp"
#include "StorageObjectBase.hpp"

namespace optifol
{

/**
 * @class ObjectGroup
 * @brief Provide a common CRTP interface for all structures grouping objects in a mutable list for iteration,
 * but also require fast lookup. The base provides a skeleton set of observing and mutating operations on the
 * model to make optimal use of the dual-storage (list and map) model.
 * @tparam Derived The IHashable type to store.
 *
 * @details
 *  The ObjectGroupBase provides two important data structures:
 *  <ul>
 *      <li>Gtk::ListStore of the type. This is a linear Gtk model that can be attached to views.</li>
 *      <li>@ref std::unordered_map of the types, mapped to their positions in the Gtk::ListStore.</li>
 *  </ul>
 *  Duplicating references across two structures provides benefits of Gtk integration through the linear
 * model, and fast lookup and hashing capability through the @ref std::unordered_map model. The models are
 * internally synchronised using the <i>libsigc++</i> callbacks provided natively by Gtkmm.
 */
template<typename Derived>
    requires std::derived_from<Derived, IHashable>
class ObjectGroup
{
public:
    ObjectGroup() = default;

    explicit ObjectGroup(sigc::slot<void(guint, guint, guint)> &&model_changed_callback)
    {
        model->signal_items_changed().connect(std::move(model_changed_callback));
```

```
}

/**
 * @brief Destruct the ObjectGroupBase.
 */
virtual ~ObjectGroup() = default;

/**
 * @brief Retrieves the untyped Gio::ListModel from a static context; this is required for GUI
 * integration.
 * @param untyped_item The Glib-enforced argument containing the object representing the object group.
 * @return The Gio::ListModel held by the object group, or the null pointer if no such model could be
 * retrieved.
 */
static Glib::RefPtr<Gio::ListModel> get_model_callback(const Glib::RefPtr<Glib::ObjectBase> &untyped_item)
{
    const auto object_group = dynamic_cast<const ObjectGroup *>(untyped_item.get());

    if (object_group != nullptr)
        return object_group->model;

    return nullptr;
}

/**
 * @brief Insert a new single object into the model.
 * @param new_object The constructed object to insert into the models.
 * @throws std::runtime_error if the new object already exists in the model.
 */
void insert_object(Glib::RefPtr<Derived> new_object)
{
    if (contains_exact(new_object.get()))
        throw std::runtime_error("Object already exists in model.");

    model->append(new_object);
    index_map[new_object] = model->get_n_items() - 1;
}

/**
 * @brief Insert multiple objects into the model.
 * @param new_objects The constructed set of objects to insert into the models.
 */
void insert_object(const std::vector<Glib::RefPtr<Derived>> &new_objects)
{
    // Append the incoming objects into the list model.
    const auto previous_count = model->get_n_items();
    model->splice(previous_count, 0, new_objects);

    // Verify insertion with the constant-time size heuristic.
    const auto new_count = model->get_n_items();
    assert(new_count == previous_count + new_objects.size());

    // Map each new object to its corresponding position in the list model.
    for (guint new_index = previous_count; new_index < new_count; ++new_index)
        index_map[new_objects[new_count - previous_count]] = new_index - 1;
}

/**
 * @brief Take a ref-counted pointer of the object at the given index.
 * @param object_index The index of the object within the linear model to take.
 * @return The requested object wrapped in a ref-counted Glib::RefPtr.
 * @throws std::runtime_error An object did not exist in the linear model at the specified index.
 */
Glib::RefPtr<Derived> get_object_by_index(guint object_index) const
{
    const auto object = model->get_item(object_index);

    if (object == nullptr)
        throw std::runtime_error("Object with given index does not exist in the Subsystem index.");

    return object;
}

/**
 * @brief Delete the given object from the model.
 * @param slated_object The object to delete.
 * @return Was an object deleted?
 */
bool delete_object(const Glib::RefPtr<Derived> slated_object)
{
    // Locate the object in the map, which will provide its index in the linear model.
```

```
    const auto index_it = index_map.find(slated_object);
    if (index_it == index_map.cend())
        return false;

    /*
     * Record as a deletion and remove from the linear model. The removal from the linear model will
     * likely trigger a user-defined callback to indicate the removal; the removed object can be retrieved
     * from the callback with 'steal_deleted_object'.
     */
    pending_deleted_items[index_it->second] = std::move(slated_object);
    model->remove(index_it->second);

    // Recalculate the index map.
    for (auto &it: index_map)
        if (it.second > index_it->second)
            --it.second;

    index_map.erase(index_it);
    return true;
}

/**
 * @brief Determines if the model contains a reference to the given object. The comparison is exact in the
 * sense that pointer addresses are compared, and does not use <code>Derived::operator==(const
 * Derived&)</code>. This is useful when working with <code>Derived</code> types adjacent to @ref
 * std::shared_ptr.
 * @param search_address The address of the item to query in the model.
 * @return Does the model already detain an item with the given address?
 */
bool contains_exact(const Derived *search_address)
{
    const auto it = index_map.find(*search_address);
    if (it == index_map.cend())
        return false;

    return it->first.get() == search_address;
}

/**
 * @brief Steal a recently deleted object from the cache. After this operation, the object is no longer
 * present in any of the models, including the deletion cache.
 * @param old_index The index of the object in the linear model prior to its deletion.
 * @return The deleted object.
 * @throws std::runtime_error if no suitable object exists in the deletion cache.
 */
Glib::RefPtr<Derived> steal_deleted_object(const guint old_index)
{
    const auto deleted_it = pending_deleted_items.find(old_index);

    if (deleted_it == pending_deleted_items.cend())
        throw std::runtime_error("Requested object at index " + std::to_string(old_index) +
                " does not exist in "
                "the deletion map.");

    const auto copy = deleted_it->second;
    pending_deleted_items.erase(deleted_it);
    return copy;
}

/**
 * @brief Populate the given Gtk::SingleSelection selection model with the internal model.
 * @param target_selection_model The selection model to populate with the current list model.
 */
void populate_selection_model(Gtk::SingleSelection &target_selection_model) const
{
    if (target_selection_model.get_model() != model)
        target_selection_model.set_model(model);
}

/**
 * @brief Populate the given Gtk::SingleSelection selection model with the internal model.
 * @param target_dropdown The Gtk::DropDown element to populate with the current list model.
 */
void populate_dropdown_model(Gtk::DropDown &target_dropdown) const
{
    if (target_dropdown.get_model() != model)
        target_dropdown.set_model(model);
}

/**
 * @brief Execute a callable over all elements of the linear model.
```

```
     * @param function The callable to execute, taking a mutating reference to the object.
     */
    void for_each(const std::function<void(Derived &)> &function) const
    {
        const auto count = model->get_n_items();
        for (guint index = 0; index < count; ++index)
            function(*model->get_item(index));
    }

    Glib::RefPtr<Gio::ListStore<Derived>> get_model() const noexcept
    {
        return model;
    }

private:
    Glib::RefPtr<Gio::ListStore<Derived>> model = Gio::ListStore<Derived>::create();

    /**
     * @brief Mapping of hashable objects present in the models, associated with their respective indices in
     * the linear
     *  @ref model.
     */
    SharedUnorderedMap<Derived, guint> index_map;

    /**
     * @brief Items deleted from the linear @ref model but not the @ref index_map. Keys indicate the old
     * positions of the values in @ref model.
     */
    std::unordered_map<guint, Glib::RefPtr<Derived>> pending_deleted_items;
};

} // namespace optifol

#endif // OBJECTGROUP_HPP
```

1.13.4 Project.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the project-level storage object
 * @author Oliver Dixon
 * @date 2025-02-22
 * @version Development
 */

#include "Project.hpp"
#include "../Logging.hpp"

namespace optifol
{

Project::Project(std::string &&name) :
    Glib::ObjectBase("Project")
{
    property_name().set_value(std::move(name));
}

Project::Project(std::string &&name, BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder) :
    Glib::ObjectBase("Project"),
    StorageObjectBase(cobject, builder)
{
    property_name().set_value(std::move(name));
}

bool Project::operator==(const Project &other) const noexcept
{
    return std::hash<Project>{}(*this) == std::hash<Project>{}(other);
}

std::string Project::get_path() const
{
    return '/' + property_name().get_value();
}
```

```
} // namespace optifol
```

1.13.5 Project.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the project-level storage object
 * @author Oliver Dixon
 * @date 2025-02-22
 * @version Development
 */

#ifndef PROJECT_HPP
#define PROJECT_HPP

#include "StorageObjectBase.hpp"
#include "TreeNode.hpp"

namespace optifol
{
/**
 * @class Project
 * @brief The Project storage forms the top level of the Optifol object hierarchy; it contains many
 * subsystems.
 */
class Project : public StorageObjectBase,
                public TreeNode
{
public:
    /**
     * @brief Create a new Project with the given name and register in the Glib GType system
     * @param name The initial name of the Project
     */
    explicit Project(std::string &&name);

    /**
     * @brief Create a new Project with the given name and register in the Glib GType system
     * @param name The initial name of the Project
     * @param cobject The C cast-item used by Glib::Object
     * @param builder Currently unused builder parameter to provide to the Glib::Object instance
     */
    Project(std::string &&name, BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);

    /**
     * @brief Tests a couple of projects for equality
     * @param other The other project
     * @return Is the current project the same as the other project?
     * @note This comparator determines equality by project metadata.
     */
    bool operator==(const Project &other) const noexcept;

    /**
     * @brief Generate a path for a root-level Project, prepended with an oblique
     * @return The project path prefix
     */
    [[nodiscard]] std::string get_path() const override;
};

} // namespace optifol

#endif
```

1.13.6 Requirement.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Requirement storage object
```

```
 * @author Oliver Dixon
 * @date 2025-04-26
 * @version Development
 */

#include "Requirement.hpp"

#include <gtkmm/label.h>
#include <gtkmm/listitem.h>

#include "../Exceptions/SemanticException.hpp"
#include "../Inference/ExpressionFactory.hpp"
#include "../Logging.hpp"
#include "../Visitors/MutableTargets/Observers/LaTeXSerialisationVisitor.hpp"
#include "../Visitors/MutableTargets/Observers/TextSerialiserVisitor.hpp"

namespace optifol
{

std::istringstream Requirement::lexer_input_stream;

FOLLexer Requirement::lexer{Requirement::lexer_input_stream, std::cerr};
FOLParser Requirement::parser{&Requirement::lexer};

const log4cxx::LoggerPtr Requirement::control_logger =
        Logging::get_logger({"GUI", "StorageControl", "Requirement"});

const log4cxx::LoggerPtr Requirement::cnf_logger = Logging::get_logger({"LogicServices", "CNFNormalisation"});
const log4cxx::LoggerPtr Requirement::parse_logger = Logging::get_logger({"LogicServices", "FormalParsing"});
const log4cxx::LoggerPtr Requirement::integration_logger =
        Logging::get_logger({"LogicServices", "SystemIntegration"});

Requirement::Requirement(std::string &&name, std::string &&statement, std::string &&description,
        const guint priority, Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &&tests, std::nullptr_t) :
    Glib::ObjectBase("Requirement"),
    statement(*this, "Requirement-statement"),
    normalised_statement(*this, "Requirement-normalised"),
    description(*this, "Requirement-description"),
    priority(*this, "Requirement-priority")
{
    setup_properties(
            std::move(name), std::move(statement), std::move(description), priority, std::move(tests));
}

Requirement::Requirement(std::string &&name, std::string &&statement, std::string &&description,
        const guint priority, Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &&tests,
        std::shared_ptr<SymbolRepository> symbol_repository) :
    Glib::ObjectBase("Requirement"),
    statement(*this, "Requirement-statement"),
    normalised_statement(*this, "Requirement-normalised"),
    description(*this, "Requirement-description"),
    priority(*this, "Requirement-priority"),
    symbol_repository(std::move(symbol_repository))
{
    setup_properties(
            std::move(name), std::move(statement), std::move(description), priority, std::move(tests));
}

Requirement::Requirement(std::string &&name, std::string &&statement, std::string &&description,
        const guint priority, Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &&tests,
        BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder,
        std::shared_ptr<SymbolRepository> symbol_repository) :
    Glib::ObjectBase("Requirement"),
    StorageObjectBase(cobject, builder),
    statement(*this, "Requirement-statement"),
    normalised_statement(*this, "Requirement-normalised"),
    description(*this, "Requirement-description"),
    priority(*this, "Requirement-priority"),
    symbol_repository(std::move(symbol_repository))
{
    setup_properties(
            std::move(name), std::move(statement), std::move(description), priority, std::move(tests));
}

Glib::RefPtr<Gtk::TreeListModel> Requirement::get_tests_tree() const noexcept
{
    return tests_tree;
}

Glib::RefPtr<Gtk::TreeListModel> Requirement::get_results_tree() const noexcept
{
```

```
    return tests_tree;
}

bool Requirement::operator==(const Requirement &other) const noexcept
{
    return hash() == other.hash();
}

Glib::PropertyProxy<Glib::ustring> Requirement::property_statement()
{
    return statement.get_proxy();
}

Glib::PropertyProxy<Glib::ustring> Requirement::property_description()
{
    return description.get_proxy();
}

Glib::PropertyProxy<guint> Requirement::property_priority()
{
    return priority.get_proxy();
}

Glib::PropertyProxy<Glib::ustring> Requirement::property_normalised()
{
    return normalised_statement.get_proxy();
}

const Gio::ListStore<Test> *Requirement::observe_tests() const noexcept
{
    return tests.get();
}

Glib::RefPtr<Gio::ListStore<Test>> Requirement::get_tests() const noexcept
{
    return tests;
}

Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> Requirement::get_test_specs() const noexcept
{
    return test_specs;
}

Glib::PropertyProxy_ReadOnly<Glib::ustring> Requirement::property_statement() const
{
    return statement.get_proxy();
}

Glib::PropertyProxy_ReadOnly<Glib::ustring> Requirement::property_description() const
{
    return description.get_proxy();
}

Glib::PropertyProxy_ReadOnly<guint> Requirement::property_priority() const
{
    return priority.get_proxy();
}

Glib::PropertyProxy_ReadOnly<Glib::ustring> Requirement::property_normalised() const
{
    return normalised_statement.get_proxy();
}

std::string Requirement::get_formatted_statement() const
{
    return formatted_input_statement;
}

std::string_view Requirement::observe_latex_statement() const noexcept
{
    return latex_input_statement;
}

bool Requirement::is_analysis_ready() const noexcept
{
    return prepared_ast != nullptr;
}

bool Requirement::has_tests() const noexcept
{
    return tests->get_n_items() > 0;
```

```
}

void Requirement::bind_name_to_label(const Glib::RefPtr<Gtk::ListItem> &item) noexcept
{
    const auto target_label = dynamic_cast<Gtk::Label *>(item->get_child());
    const auto typed_group = dynamic_cast<const Requirement *>(item->get_item().get());
    if (target_label == nullptr || typed_group == nullptr)
        return;

    target_label->set_text(typed_group->property_name().get_value());
}

const SentenceRoot &Requirement::observe_prepared_sentence() const
{
    if (!is_analysis_ready())
        throw std::runtime_error("The Requirement is not ready for FOL analysis.");

    return *prepared_ast;
}

void Requirement::setup_properties(std::string &&requirement_name, std::string &&requirement_statement,
        std::string &&requirement_description, const guint requirement_priority,
        Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &&requirement_tests)
{
    property_statement().signal_changed().connect(
            sigc::mem_fun(*this, &Requirement::handle_statement_change));

    property_name().set_value(std::move(requirement_name));
    property_statement().set_value(std::move(requirement_statement));
    property_description().set_value(std::move(requirement_description));
    property_priority().set_value(requirement_priority);
    test_specs = std::move(requirement_tests);

    test_specs->signal_items_changed().connect(sigc::mem_fun(*this, &Requirement::handle_test_spec_change));
    handle_test_spec_change(0, 0, test_specs->get_n_items()); // On first setup, all items are new.
}

void Requirement::handle_statement_change()
{
    // If the statement has changed, pass it through the parser and normaliser.
    const auto &typed_statement = property_statement().get_value();

    if (typed_statement.empty() == true) {
        original_ast.reset();
        formatted_input_statement.clear();
        latex_input_statement.clear();
        prepared_ast.reset();

        control_logger->debug(
                "Removed FOL statement from requirement \"" + property_name().get_value() + "\".");
        return;
    }

    lexer_input_stream.str(typed_statement);

    try {
        parser.parse();
    } catch (const ParseError &parse_error) {
        parse_logger->error(parse_error.what());
        return;
    }

    original_ast = parser.retrieve_sentence();
    formatted_input_statement = text_serialise(original_ast.get());
    latex_input_statement = latex_serialise(original_ast.get());

    try {
        // Perform CNF normalisation followed by population of the symbol repository
        prepared_ast = ExpressionFactory::build_sentence(original_ast->clone_as_root(), symbol_repository);
    } catch (const SemanticException &) {
        cnf_logger->error("Preparation process was unsuccessful due to invalid logical semantics; "
                          "requirements will be missing.");
        return;
    }

    std::ostringstream serialiser_stream;
    prepared_ast->serialise(serialiser_stream);
    property_normalised().set_value(serialiser_stream.str());

    control_logger->debug(
            "Successfully updated FOL statement for requirement \"" + property_name().get_value() + "\".");
```

```
}

void Requirement::handle_test_spec_change(
        const guint position, const guint removed_count, const guint added_count) const
{
    std::vector<Glib::RefPtr<Test>> new_tests;
    new_tests.reserve(added_count);

    for (guint spec_index = position; spec_index < added_count; ++spec_index) {
        const auto &spec = test_specs->get_item(spec_index);
        if (spec == nullptr || spec->property_executable().get_value() == nullptr)
            control_logger->warn(
                    "Ignoring invalid test specification at index " + std::to_string(spec_index));
        else
            new_tests.push_back(Glib::make_refptr_for_instance(new Test(spec)));
    }

    tests->splice(position, removed_count, new_tests);
}

std::string Requirement::text_serialise(const IMutableSentence *sentence)
{
    TextSerialiserVisitor text_serialiser_visitor;
    sentence->accept(text_serialiser_visitor);
    return text_serialiser_visitor.extract();
}

std::string Requirement::latex_serialise(const IMutableSentence *sentence)
{
    LaTeXSerialisationVisitor latex_serialisation_visitor;
    sentence->accept(latex_serialisation_visitor);
    return latex_serialisation_visitor.extract();
}

} // namespace optifol
```

1.13.7 Requirement.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Requirement storage object
 * @author Oliver Dixon
 * @date 2025-04-26
 * @version Development
 */

#ifndef REQUIREMENT_HPP
#define REQUIREMENT_HPP

#include <gtkmm/treelistmodel.h>
#include <log4cxx/logger.h>

#include "../IR/MutableVariants/Sentences/IMutableSentence.hpp"
#include "../IR/Sentences/SentenceRoot.hpp"
#include "../IR/SymbolRepository.hpp"
#include "../UserTesting/Modelling/Test.hpp"
#include "../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "FOLLexer.hpp"
#include "StorageObjectBase.hpp"

namespace Gtk
{

class ListItem;
class Label;

} // namespace Gtk

namespace optifol
{

/**
 * @class Requirement
 * @brief The Requirement storage object is the atomic unit of measure in Optifol. It belongs to a single
```

```
 * Subsystem.
 */
class Requirement : public StorageObjectBase,
                    public ITestModelNode
{
public:
    /**
     * @brief Create a new Requirement with the given name and register in the Glib GType system,
     *  but no active SymbolRepository.
     * @param name The initial name of the Requirement
     * @param statement The initial FOL statement of the Requirement
     * @param description The initial long-form description of the Requirement
     * @param priority The initial priority of the Requirement
     * @param tests The initial set of TestSpecificationEntry objects to template Test objects.
     * @param symbol_repository A null pointer to explicitly signify the lacking SymbolRepository
     * @warning As no system-wide symbol repository has been provided, this Requirement will not supply its
     * symbols to the wider system. Logical analysis will produce unexpected results.
     */
    explicit Requirement(std::string &&name, std::string &&statement, std::string &&description,
            guint priority, Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &&tests,
            std::nullptr_t symbol_repository);

    /**
     * @brief Create a new Requirement with the given name and register in the Glib GType system
     * @param name The initial name of the Requirement
     * @param statement The initial FOL statement of the Requirement
     * @param description The initial long-form description of the Requirement
     * @param priority The initial priority of the Requirement
     * @param tests The initial set of TestSpecificationEntry objects to template Test objects.
     * @param symbol_repository The system-wide symbol repository with lifetimes guaranteed to cover that of
     * the Requirement
     */
    explicit Requirement(std::string &&name, std::string &&statement, std::string &&description,
            guint priority, Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &&tests,
            std::shared_ptr<SymbolRepository> symbol_repository);

    /**
     * @brief Create a new Requirement with the given name and register in the Glib GType system
     * @param name The initial name of the Subsystem
     * @param statement The initial FOL statement of the Requirement
     * @param description The initial long-form description of the Requirement
     * @param priority The initial priority of the Requirement
     * @param tests The initial set of TestSpecificationEntry objects to template Test objects.
     * @param cobject The C cast-item used by Glib::Object
     * @param builder Currently unused builder parameter to provide to the Glib::Object instance
     * @param symbol_repository The system-wide symbol repository with lifetimes guaranteed to cover that of
     * the Requirement
     */
    Requirement(std::string &&name, std::string &&statement, std::string &&description, guint priority,
            Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &&tests, BaseObjectType *cobject,
            const Glib::RefPtr<Gtk::Builder> &builder, std::shared_ptr<SymbolRepository> symbol_repository);

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_tests_tree() const noexcept override;

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_results_tree() const noexcept override;

    /**
     * @brief Compare two Requirement objects for semantic equality
     * @param other The Requirement with which to compare.
     * @return Is the current Requirement equivalent to the given other Requirement?
     */
    bool operator==(const Requirement &other) const noexcept;

    /**
     * @brief Get a read-write proxy for the 'statement' property
     * @return The read-write 'statement' proxy
     */
    [[nodiscard]] Glib::PropertyProxy<Glib::ustring> property_statement();

    /**
     * @brief Get a read-write proxy for the 'description' property
     * @return The read-write 'description' proxy
     */
    [[nodiscard]] Glib::PropertyProxy<Glib::ustring> property_description();

    /**
     * @brief Get a read-write proxy for the 'priority' property
     * @return The read-write 'priority' proxy
     */
    [[nodiscard]] Glib::PropertyProxy<guint> property_priority();
```

```
    /**
     * @brief Get a read-write proxy for the 'normalised statement' property
     * @return The read-write 'normalised statement' proxy
     */
    [[nodiscard]] Glib::PropertyProxy<Glib::ustring> property_normalised();

    [[nodiscard]] const Gio::ListStore<Test> *observe_tests() const noexcept;

    [[nodiscard]] Glib::RefPtr<Gio::ListStore<Test>> get_tests() const noexcept;

    [[nodiscard]] Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> get_test_specs() const noexcept;

    /**
     * @brief Get a read-only proxy for the 'statement' property
     * @return The read-only 'statement' proxy
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::ustring> property_statement() const;

    /**
     * @brief Get a read-only proxy for the 'description' property
     * @return The read-only 'description' proxy
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::ustring> property_description() const;

    /**
     * @brief Get a read-only proxy for the 'priority' property
     * @return The read-only 'priority' proxy
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<guint> property_priority() const;

    /**
     * @brief Get a read-only proxy for the 'normalised statement' property
     * @return The read-only 'normalised statement' proxy
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::ustring> property_normalised() const;

    [[nodiscard]] std::string get_formatted_statement() const;

    [[nodiscard]] std::string_view observe_latex_statement() const noexcept;

    /**
     * @brief Determines the suitability of the Requirement for participation in formal analysis.
     * @return Does the Requirement have a prepared and normalised statement AST?
     */
    [[nodiscard]] bool is_analysis_ready() const noexcept;

    /**
     * @brief Determines whether the Requirement has at least one associated Test. This is commonly useful for
     *  determining its suitability to be added to a TestGroup.
     * @return Does the Requirement have at least one Test?
     */
    [[nodiscard]] bool has_tests() const noexcept;

    static void bind_name_to_label(const Glib::RefPtr<Gtk::ListItem> &item) noexcept;

    const SentenceRoot &observe_prepared_sentence() const;

private:
    static const log4cxx::LoggerPtr control_logger;
    static const log4cxx::LoggerPtr parse_logger;
    static const log4cxx::LoggerPtr cnf_logger;
    static const log4cxx::LoggerPtr integration_logger;

    /**
     * @brief Helper to push the given IMutableSentence node through a plain-text serialisation pipeline
     * @param sentence The sentence to serialise
     * @return The @ref std::string representation of the plain-text serialised sentence
     */
    static std::string text_serialise(const IMutableSentence *sentence);

    /**
     * @brief Helper to push the given IMutableSentence node through a LaTeX-text serialisation pipeline
     * @param sentence The sentence to serialise into math-mode LaTeX format
     * @return The @ref std:string representation of the LaTeX-escaped math-mode serialised sentence
     */
    static std::string latex_serialise(const IMutableSentence *sentence);

    /**
     * @brief Populate the core Requirement properties with the given initial values, and set up signals.
     * @param requirement_name The initial Requirement name
     * @param requirement_statement The initial Requirement statement text
```

```
     * @param requirement_description The initial Requirement description
     * @param requirement_priority The initial Requirement priority selection
     * @param requirement_tests The initial set of TestSpecificationEntry objects to template Test objects.
     */
    void setup_properties(std::string &&requirement_name, std::string &&requirement_statement,
            std::string &&requirement_description, guint requirement_priority,
            Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> &&requirement_tests);

    /**
     * @brief Handle a change in the Requirement statement by re-parsing and updating internal state where
     * necessary.
     */
    void handle_statement_change();

    void handle_test_spec_change(guint position, guint removed_count, guint added_count) const;

    Glib::Property<Glib::ustring> statement;
    Glib::Property<Glib::ustring> normalised_statement;
    Glib::Property<Glib::ustring> description;
    Glib::Property<guint> priority;

    Glib::RefPtr<Gio::ListStore<TestSpecificationEntry>> test_specs =
            Gio::ListStore<TestSpecificationEntry>::create();
    const Glib::RefPtr<Gio::ListStore<Test>> tests = Gio::ListStore<Test>::create();
    const Glib::RefPtr<Gtk::TreeListModel> tests_tree =
            Gtk::TreeListModel::create(tests, &ITestModelNode::get_given_tests_tree, true);

    std::unique_ptr<MutableSentenceRoot> original_ast;
    std::unique_ptr<SentenceRoot> prepared_ast;
    std::shared_ptr<SymbolRepository> symbol_repository;

    std::string formatted_input_statement;
    std::string latex_input_statement;

    static std::istringstream lexer_input_stream;
    static FOLLexer lexer;
    static FOLParser parser;
};

} // namespace optifol

#endif
```

1.13.8 StorageObjectBase.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the base class of a Storable Object
 * @author Oliver Dixon
 * @date 2025-04-26
 * @version Development
 */

#include <glibmm/binding.h>
#include <gtkmm/label.h>
#include <gtkmm/listitem.h>
#include <gtkmm/treeexpander.h>
#include <gtkmm/treelistmodel.h>

#include "StorageObjectBase.hpp"

namespace optifol
{

Glib::PropertyProxy<Glib::ustring> StorageObjectBase::property_name()
{
    return name.get_proxy();
}

Glib::PropertyProxy<StorageObjectBase::TimeT> StorageObjectBase::property_creation_time()
{
    return creation_time.get_proxy();
}
```

```
Glib::PropertyProxy<StorageObjectBase::TimeT> StorageObjectBase::property_modified_time()
{
    return modified_time.get_proxy();
}

Glib::PropertyProxy_ReadOnly<Glib::ustring> StorageObjectBase::property_name() const
{
    return name.get_proxy();
}

Glib::PropertyProxy_ReadOnly<StorageObjectBase::TimeT> StorageObjectBase::property_creation_time() const
{
    return creation_time.get_proxy();
}

Glib::PropertyProxy_ReadOnly<StorageObjectBase::TimeT> StorageObjectBase::property_modified_time() const
{
    return modified_time.get_proxy();
}

std::size_t StorageObjectBase::hash() const noexcept
{
    return std::hash<std::string>{}(property_name().get_value());
}

void StorageObjectBase::bind_name(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
    const auto item = std::dynamic_pointer_cast<StorageObjectBase>(list_item->get_item());

    if (label != nullptr && item != nullptr)
        Glib::Binding::bind_property(
                item->property_name(), label->property_label(), Glib::Binding::Flags::SYNC_CREATE);
}

void StorageObjectBase::bind_creation_time(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
    const auto item = std::dynamic_pointer_cast<StorageObjectBase>(list_item->get_item());

    if (label != nullptr && item != nullptr)
        Glib::Binding::bind_property(item->property_creation_time(), label->property_label(),
                Glib::Binding::Flags::SYNC_CREATE,
                [](const std::chrono::system_clock::time_point &time) { return std::format("{:%c}", time); });
}

void StorageObjectBase::bind_modification_time(const Glib::RefPtr<Gtk::ListItem> &list_item)
{
    const auto label = dynamic_cast<Gtk::Label *>(list_item->get_child());
    const auto item = std::dynamic_pointer_cast<StorageObjectBase>(list_item->get_item());

    if (label != nullptr && item != nullptr)
        Glib::Binding::bind_property(item->property_modified_time(), label->property_label(),
                Glib::Binding::Flags::SYNC_CREATE,
                [](const std::chrono::system_clock::time_point &time) { return std::format("{:%c}", time); });
}

void StorageObjectBase::bind_name_property_expandable(const Glib::RefPtr<Gtk::ListItem> &list_item,
        const Glib::RefPtr<Gtk::TreeListModel> &tree_model) noexcept
{
    const auto position = list_item->get_position();
    const auto model_item = std::dynamic_pointer_cast<StorageObjectBase>(list_item->get_item());
    const auto expander = dynamic_cast<Gtk::TreeExpander *>(list_item->get_child());

    if (position == GTK_INVALID_LIST_POSITION || model_item == nullptr || expander == nullptr)
        return;

    const auto gui_row = tree_model->get_row(position);
    if (!gui_row)
        return;

    expander->set_list_row(gui_row);

    const auto label = dynamic_cast<Gtk::Label *>(expander->get_child());
    if (!label)
        return;

    Glib::Binding::bind_property(
            model_item->property_name(), label->property_label(), Glib::Binding::Flags::SYNC_CREATE);
}
```

```
StorageObjectBase::StorageObjectBase() :
    Glib::ObjectBase("StorageObjectBase"),
    name(*this, "StorageObjectBase-name"),
    creation_time(*this, "StorageObjectBase-creation-time", std::chrono::system_clock::now()),
    modified_time(*this, "StorageObjectBase-modified-time", std::chrono::system_clock::now())
{
}

StorageObjectBase::StorageObjectBase(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &) :
    Glib::ObjectBase("StorageObjectBase"),
    Glib::Object(cobject),
    name(*this, "StorageObjectBase-name"),
    creation_time(*this, "StorageObjectBase-creation-time", std::chrono::system_clock::now()),
    modified_time(*this, "StorageObjectBase-modified-time", std::chrono::system_clock::now())
{
}

} // namespace optifol
```

1.13.9 StorageObjectBase.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the base class of a Storable Object
 * @author Oliver Dixon
 * @date 2025-02-22
 * @version Development
 */

#ifndef STORAGEOBJECTBASE_HPP
#define STORAGEOBJECTBASE_HPP

#include <chrono>
#include <glibmm/object.h>
#include <glibmm/property.h>

#include "../IHashable.hpp"

namespace Gtk
{

class TreeListModel;
class ListItem;
class Label;
class Builder;

} // namespace Gtk

namespace optifol
{

/**
 * @class StorageObjectBase
 * @brief Defines the base class for a Glib-registered storable object in the Optifol storage hierarchy
 */
class StorageObjectBase : public Glib::Object,
                          public IHashable
{
public:
    /**
     * @typedef TimeT
     * @brief The internal C++ type used to manipulate times on the system clock
     */
    using TimeT = std::chrono::system_clock::time_point;

    [[nodiscard]] std::size_t hash() const noexcept override;

    /**
     * @brief Get a read-write proxy for the 'name' property
     * @return The read-write 'name' proxy
     */
    [[nodiscard]] Glib::PropertyProxy<Glib::ustring> property_name();

    /**
```

```
     * @brief Get a read-write proxy for the 'creation time' property
     * @return The read-write 'creation time' proxy
     */
    [[nodiscard]] Glib::PropertyProxy<TimeT> property_creation_time();

    /**
     * @brief Get a read-write proxy for the 'last-modified time' property
     * @return The read-write 'last-modified time' proxy
     */
    [[nodiscard]] Glib::PropertyProxy<TimeT> property_modified_time();

    /**
     * @brief Get a read-only proxy for the 'name' property
     * @return The read-only 'name' proxy
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::ustring> property_name() const;

    /**
     * @brief Get a read-only proxy for the 'creation time' property
     * @return The read-only 'creation time' proxy
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<TimeT> property_creation_time() const;

    /**
     * @brief Get a read-only proxy for the 'last-modified time' property
     * @return The read-only 'last-modified time' proxy
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<TimeT> property_modified_time() const;

    /**
     * @brief Establish a property-synched binding between the 'name' property of a StorableObjectBase object,
     * and a flat (non-expandable) label in a Gtk::ListView.
     * @param list_item The list item provided by the GTK callback invocation
     */
    static void bind_name(const Glib::RefPtr<Gtk::ListItem> &list_item);

    /**
     * @brief Establish a property-synched binding between the 'creation time' property of a
     * StorableObjectBase object, and a flat (non-expandable) label in a Gtk::ListView by means of a
     * locale-dependent formatting routine.
     * @param list_item The list item provided by the GTK callback invocation
     */
    static void bind_creation_time(const Glib::RefPtr<Gtk::ListItem> &list_item);

    /**
     * @brief Establish a property-synched binding between the 'last-modified time' property of a
     * StorableObjectBase- like object, and a flat (non-expandable) label in a Gtk::ListView by means of a
     * locale-dependent formatting routine.
     * @param list_item The list item provided by the GTK callback invocation
     */
    static void bind_modification_time(const Glib::RefPtr<Gtk::ListItem> &list_item);

    /**
     * @brief Establish a property-synched binding between the 'name' property of a StorableObjectBase-like
     * object, and a tree-expandable label in a Gtk::ListView with nested expanders.
     * @param list_item The list item provided by the GTK callback invocation
     * @param tree_model The tree model in which the list item exists, required to update expander
     * responsibility delegation
     */
    static void bind_name_property_expandable(const Glib::RefPtr<Gtk::ListItem> &list_item,
            const Glib::RefPtr<Gtk::TreeListModel> &tree_model) noexcept;

protected:
    /**
     * @brief Derive from the common storage base with the universal of GType-registered properties
     */
    StorageObjectBase();

    /**
     * @brief Derive from the common storage base with the universal of GType-registered properties
     * @param cobject The C cast-item used by Glib::Object
     * @param builder Currently unused builder parameter to provide to the Glib::Object instance
     */
    StorageObjectBase(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);

private:
    Glib::Property<Glib::ustring> name;

    Glib::Property<TimeT> creation_time;

    Glib::Property<TimeT> modified_time;
```

```
};


// Current Clang 18 bug reports Doxygen violations for uses of @tparam on templated concepts.
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wdocumentation"
/**
 * @concept StorableType
 * @brief Represents a type that is declared to be a permanently storable object in the Optifol type system
 * @tparam Type The implementing type of the storable object
 */
template<typename Type>
concept StorableType = std::derived_from<Type, StorageObjectBase>;
#pragma clang diagnostic pop

} // namespace optifol

#endif
```

1.13.10 Subsystem.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the subsystem-level storage object
 * @author Oliver Dixon
 * @date 2025-02-22
 * @version Development
 */

#include <cassert>

#include "../Logging.hpp"
#include "Subsystem.hpp"

namespace optifol
{

const log4cxx::LoggerPtr Subsystem::subsystem_logger =
        Logging::get_logger({"GUI", "StorageControl", "Subsystem"});

Subsystem::Subsystem(const Glib::ustring &name, TreeNode *parent) :
    Glib::ObjectBase("Subsystem"),
    TreeNode(parent),
    ObjectGroup(sigc::mem_fun(*this, &Subsystem::handle_requirement_model_change))
{
    setup_groups(name);
}

Subsystem::Subsystem(const Glib::ustring &name, BaseObjectType *cobject,
        const Glib::RefPtr<Gtk::Builder> &builder, TreeNode *parent) :
    Glib::ObjectBase("Subsystem"),
    StorageObjectBase(cobject, builder),
    TreeNode(parent),
    ObjectGroup(sigc::mem_fun(*this, &Subsystem::handle_requirement_model_change))
{
    setup_groups(name);
}

bool Subsystem::operator==(const Subsystem &other) const noexcept
{
    return hash() == other.hash();
}

std::string Subsystem::get_path() const
{
    const auto hash = std::hash<Subsystem>{}(*this);
    if (hash != fully_qualified_path_cache.first) {
        fully_qualified_path_cache.first = hash;
        fully_qualified_path_cache.second = get_parent()->get_path() + '/' + property_name().get_value();
    }

    return fully_qualified_path_cache.second;
}
```

```
void Subsystem::duplicate_requirement(const Requirement &requirement)
{
    build_requirement(requirement.property_name().get_value(), requirement.property_statement().get_value(),
            requirement.property_description().get_value(), requirement.property_priority().get_value(),
            requirement.get_test_specs());
}

Glib::RefPtr<Gio::ListStore<AnalysisGroup>> Subsystem::get_analysis_groups() const noexcept
{
    return analysis_groups;
}

Glib::RefPtr<Gio::ListStore<TestGroup>> Subsystem::get_test_groups() const noexcept
{
    return test_groups;
}

std::shared_ptr<SymbolRepository> Subsystem::share_symbol_repository()
{
    return symbol_repository;
}

void Subsystem::setup_groups(const Glib::ustring &name)
{
    assert(analysis_groups->get_n_items() == 0);
    assert(test_groups->get_n_items() == 0);

    property_name().set_value(name);

    analysis_groups->append(
            Glib::make_refptr_for_instance(new AnalysisGroup("Unassigned Requirements", symbol_repository)));

    test_groups->append(Glib::make_refptr_for_instance(new TestGroup("Unassigned Requirements")));

    assert(analysis_groups->get_n_items() == 1);
    assert(test_groups->get_n_items() == 1);
}

void Subsystem::handle_requirement_model_change(
        const guint initial_index, const guint removed_count, const guint added_count) noexcept
{
    subsystem_logger->debug("Handling requirements change: " + std::to_string(added_count) +
            " additions and " + std::to_string(removed_count) + " deletions at position " +
            std::to_string(initial_index) + '.');

    handle_requirement_deletions(initial_index, removed_count);
    handle_requirement_additions(initial_index, added_count);
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive. Called from 'handle_object_change'.
void Subsystem::handle_requirement_deletions(const guint initial_index, const guint removed_count) noexcept
{
    /*
     * First handle removed requirements, making use of the deleted requirements records. Notice that this
     * doesn't use the model at all, as the Requirements have already been erased and moved into the deleted
     * records map.
     */

    for (guint remove_count_i = 0; remove_count_i < removed_count; ++remove_count_i) {
        try {
            const auto deleted_item = steal_deleted_object(initial_index + remove_count_i);

            // Remove from analysis groups.
            if (deleted_item->is_analysis_ready()) {
                const auto analysis_group_count = analysis_groups->get_n_items();

                for (guint analysis_group_index = 0; analysis_group_index < analysis_group_count;
                        ++analysis_group_index) {
                    const auto &analysis_group = analysis_groups->get_item(analysis_group_index);
                    subsystem_logger->debug("Propagating deletion of \"" +
                            deleted_item->property_name().get_value() + "\" to Analysis Group \"" +
                            analysis_group->property_name().get_value() + "\".");
                    analysis_group->delete_object(deleted_item);
                }
            }

            // Remove from test groups.
            if (deleted_item->has_tests()) {
                const auto test_group_count = test_groups->get_n_items();

                for (guint test_group_index = 0; test_group_index < test_group_count; ++test_group_index) {
```

```
                const auto &test_group = test_groups->get_item(test_group_index);
                subsystem_logger->debug("Propagating deletion of \"" +
                        deleted_item->property_name().get_value() + "\" to Test Group \"" +
                        test_group->property_name().get_value() + "\".");
                test_group->delete_object(deleted_item);
            }
        }
    } catch (const std::runtime_error &error) {
        subsystem_logger->error("Could not propagate deletion of Requirement previously at index " +
                std::to_string(initial_index + remove_count_i) + " through Subsystem \"" +
                property_name().get_value() + "\".");
        subsystem_logger->error(error.what());
    }
  }
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive. Called from 'handle_object_change'.
void Subsystem::handle_requirement_additions(
        const guint initial_index, const guint added_count) const noexcept
{
    const auto default_analysis_group = analysis_groups->get_item(0);
    const auto default_test_group = test_groups->get_item(0);

    if (added_count == 1) {
        try {
            const auto candidate = get_object_by_index(initial_index);

            // Distribute to default analysis group, if it has an OK formula.
            if (candidate->is_analysis_ready()) {
                subsystem_logger->debug("Propagating addition of \"" +
                        candidate->property_name().get_value() + "\" to default Analysis Group.");
                default_analysis_group->insert_object(candidate);
            }

            // Distribute to default test group, if it has at least one test.
            if (candidate->has_tests()) {
                subsystem_logger->debug("Propagating addition of \"" +
                        candidate->property_name().get_value() + "\" to default Test Group.");
                default_test_group->insert_object(candidate);
            }
        } catch (const std::runtime_error &error) {
            subsystem_logger->error("Could not propagate addition of Requirement claimed to be at index " +
                    std::to_string(initial_index) + " through Subsystem \"" + property_name().get_value() +
                    "\".");
            subsystem_logger->error(error.what());
        }
    }

    else if (added_count > 1) {
        // As above, but specialised for multiple insertion using Gtk's splice capability.

        try {
            std::vector<Glib::RefPtr<Requirement>> analysis_additions;
            std::vector<Glib::RefPtr<Requirement>> testing_additions;

            analysis_additions.reserve(added_count);
            testing_additions.reserve(added_count);

            for (guint added_list_i = initial_index; added_list_i < added_count; ++added_list_i) {
                try {
                    const auto candidate = get_object_by_index(added_list_i + initial_index);

                    if (candidate->is_analysis_ready()) {
                        subsystem_logger->debug("Propagating addition of \"" +
                                candidate->property_name().get_value() + "\" to default Analysis Group.");
                        analysis_additions.push_back(candidate);
                    }

                    if (candidate->has_tests()) {
                        subsystem_logger->debug("Propagating addition of \"" +
                                candidate->property_name().get_value() + "\" to default Test Group.");
                        testing_additions.push_back(candidate);
                    }
                } catch (const std::runtime_error &error) {
                    subsystem_logger->error(
                            "Could not propagate addition of Requirement claimed to be at index " +
                            std::to_string(initial_index) + " through Subsystem \"" +
                            property_name().get_value() + "\".");
                    subsystem_logger->error(error.what());
                }
            }
```

```
            default_analysis_group->insert_object(analysis_additions);
            default_test_group->insert_object(testing_additions);
        } catch (const std::exception &global_error) {
            subsystem_logger->error("Could not propagate any additions of " + std::to_string(added_count) +
                    " from index " + std::to_string(initial_index) + " for Subsystem " +
                    property_name().get_value() + "\".");
            subsystem_logger->error(global_error.what());
        }
    }
}

} // namespace optifol
```

1.13.11 Subsystem.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Subsystem-level storage object
 * @author Oliver Dixon
 * @date 2025-02-22
 * @version Development
 */

#ifndef SUBSYSTEM_HPP
#define SUBSYSTEM_HPP

#include <giomm/liststore.h>
#include <gtkmm/singleselection.h>

#include "../UserTesting/Modelling/TestGroup.hpp"
#include "AnalysisGroup.hpp"
#include "Requirement.hpp"
#include "StorageObjectBase.hpp"
#include "TreeNode.hpp"

namespace optifol
{

/**
 * @class Subsystem
 * @brief The Subsystem storage object forms the second level of the Optifol object hierarchy; it belongs to a
 * Project, and consists of many individual requirements.
 * @details The Subsystem holds a great amount of responsibility. It is the single-owning repository for much
 * of the storage hierarchy. In particular, it creates and persists the following objects: <ul>
 *      <li>Requirement objects in the flat structure;</li>
 *      <li>AnalysisGroup objects; and</li>
 *      <li>TestGroup objects.</li>
 *  </ul>
 *  Therefore to handle changes in its base set of Requirement objects, changes must be propagated to all
 * relevant AnalysisGroup and TestGroup grouping structures.
 */
class Subsystem : public StorageObjectBase,
                  public TreeNode,
                  public ObjectGroup<Requirement>
{
public:
    /**
     * @brief Create a new Subsystem with the given name and register in the Glib GType system
     * @param name The initial name of the Subsystem
     * @param parent The owning node: typically a Project (if root-level Subsystem) or Subsystem if a member
     * of a nested hierarchy.
     */
    explicit Subsystem(const Glib::ustring &name, TreeNode *parent);

    /**
     * @brief Create a new Subsystem with the given name and register in the Glib GType system
     * @param name The initial name of the Subsystem
     * @param cobject The C cast-item used by Glib::Object
     * @param builder Currently unused builder parameter to provide to the Glib::Object instance
     * @param parent The owning node: typically a Project (if root-level Subsystem) or Subsystem if a member
     * of a nested hierarchy.
     */
    Subsystem(const Glib::ustring &name, BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder,
```

```
            TreeNode *parent);

    /**
     * @brief Tests a couple of subsystem for equality
     * @param other The other subsystem
     * @return Is the current subsystem the same as the other subsystem?
     * @note This comparator determines equality by subsystem metadata.
     */
    bool operator==(const Subsystem &other) const noexcept;

    /**
     * @brief Recursively generate a human-readable path of the Subsystem hierarchy, delimited with oblique
     * characters
     * @return The human-readable path of the current subsystem with a leading oblique
     */
    [[nodiscard]] std::string get_path() const override;

    /**
     * @brief Build a new Requirement, using the forwarded arguments, with the Subsystem SymbolRepository
     * instance. The built Requirement is immediately appended to the model.
     * @tparam CtorArgs The types of Requirement constructor arguments to forward, as a parameter pack.
     * @param args The head argument values to forward to the Requirement constructor.
     */
    template<typename... CtorArgs>
    void build_requirement(CtorArgs &&...args)
    {
        insert_object(Glib::make_refptr_for_instance(
                new Requirement(std::forward<CtorArgs>(args)..., symbol_repository)));
    }

    /**
     * @brief Duplicate the given Requirement and append to the index.
     * @param requirement The Requirement to duplicate.
     * @throws std::runtime_error if the duplicated Requirement could not be created.
     */
    void duplicate_requirement(const Requirement &requirement);

    Glib::RefPtr<Gio::ListStore<AnalysisGroup>> get_analysis_groups() const noexcept;

    Glib::RefPtr<Gio::ListStore<TestGroup>> get_test_groups() const noexcept;

    std::shared_ptr<SymbolRepository> share_symbol_repository();

private:
    /**
     * @brief Configure the Subsystem to a known initial state, including the configuration of signal handlers
     * for changing internal list models, and construction of default test and analysis groups.
     * @param name The initial name of the Subsystem
     */
    void setup_groups(const Glib::ustring &name);

    void handle_requirement_model_change(
            guint initial_index, guint removed_count, guint added_count) noexcept;

    /**
     * @brief Propagate Requirement model deletions to the analysis and test groups.
     * @param initial_index The index at which the deletion started.
     * @param removed_count The number of deletions from the initial index.
     */
    void handle_requirement_deletions(guint initial_index, guint removed_count) noexcept;

    /**
     * @brief Propagate Requirement model additions to the analysis and test groups.
     * @param initial_index The index at which the addition started.
     * @param added_count The number of additions from the initial index.
     */
    void handle_requirement_additions(guint initial_index, guint added_count) const noexcept;

    static const log4cxx::LoggerPtr subsystem_logger;

    Glib::RefPtr<Gio::ListStore<AnalysisGroup>> analysis_groups = Gio::ListStore<AnalysisGroup>::create();
    Glib::RefPtr<Gio::ListStore<TestGroup>> test_groups = Gio::ListStore<TestGroup>::create();

    std::shared_ptr<SymbolRepository> symbol_repository = std::make_shared<SymbolRepository>();

    mutable std::pair<std::size_t, std::string> fully_qualified_path_cache;
};

} // namespace optifol

#endif
```

1.13.12 TreeNode.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the tree node abstract storage object
 * @author Oliver Dixon
 * @date 2025-04-26
 * @version Development
 */

#include "TreeNode.hpp"

#include "Subsystem.hpp"

namespace optifol
{

TreeNode::TreeNode() :
    children(Gio::ListStore<Subsystem>::create())
{
}

TreeNode::TreeNode(TreeNode *const parent) :
    children(Gio::ListStore<Subsystem>::create()),
    parent(parent)
{
}

Glib::RefPtr<Gio::ListStore<Subsystem>> TreeNode::get_children()
{
    return children;
}

Glib::RefPtr<const Gio::ListStore<Subsystem>> TreeNode::get_children() const
{
    return children;
}

void TreeNode::add(std::string &&name)
{
    children->append(Glib::make_refptr_for_instance(new Subsystem(std::move(name), this)));
}

const TreeNode *TreeNode::get_parent() const
{
    return parent;
}

TreeNode *TreeNode::get_parent()
{
    return parent;
}

} // namespace optifol
```

1.13.13 TreeNode.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the tree node abstract storage object
 * @author Oliver Dixon
 * @date 2025-04-26
 * @version Development
 */

#ifndef TREENODE_HPP
#define TREENODE_HPP

#include <chrono>
```

```cpp
#include <giomm/liststore.h>

namespace optifol
{

class Subsystem;

/**
 * @class TreeNode
 * @brief The TreeNode is an abstract storage object to aid in the nesting of subsystems to arbitrary depths.
 */
class TreeNode
{
public:
    /**
     * @brief Destruct the TreeNode
     */
    virtual ~TreeNode() = default;

    [[nodiscard]] Glib::RefPtr<Gio::ListStore<Subsystem>> get_children();

    [[nodiscard]] Glib::RefPtr<const Gio::ListStore<Subsystem>> get_children() const;

    void add(std::string &&name);

    [[nodiscard]] const TreeNode *get_parent() const;

    [[nodiscard]] TreeNode *get_parent();

    [[nodiscard]] virtual std::string get_path() const = 0;

protected:
    /**
     * @brief Create a new TreeNode without a parent
     */
    TreeNode();

    /**
     * @brief Create a new TreeNode with a parent
     * @param parent A weak pointer to the parent
     */
    explicit TreeNode(TreeNode *parent);

private:
    Glib::RefPtr<Gio::ListStore<Subsystem>> children;

    TreeNode *const parent = nullptr;
};

} // namespace optifol

#endif
```

1.14 Tests

1.14.1 BidirectionalUnificationTest.cpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Test bidirectional unification of FOL sentences and terms
 * @author Oliver Dixon
 * @date 2025-06-08
 * @version Development
 */

#include <gtest/gtest.h>

#include "../IR/Sentences/Literal.hpp"
#include "../IR/SymbolRepository.hpp"
#include "../IR/Terms/Function.hpp"
#include "../IR/Terms/Variable.hpp"
#include "../Visitors/RegularTargets/Unification/BidirectionalUnificationVisitor.hpp"

namespace optifol
```

```
{

/**
 * @class BidirectionalUnificationTest
 * @brief Tests the BidirectionalUnificationTest to verify that valid substitutions are constructed in the
 *  Generalisations and Instances to unify pairs of sentences.
 */
class BidirectionalUnificationTest : public testing::Test
{
protected:
    std::unique_ptr<BidirectionalUnificationVisitor> unification_visitor;

    void SetUp() override
    {
        symbol_repository = std::make_shared<SymbolRepository>();
        unification_visitor = std::make_unique<BidirectionalUnificationVisitor>(symbol_repository);
    }

    template<typename TermType, class... CtorArgs>
    [[nodiscard]] const TermType *register_symbol(CtorArgs &&...ctor_args) const
    {
        return symbol_repository->add_symbol(
                std::make_unique<TermType>(std::forward<CtorArgs>(ctor_args)...));
    }

private:
    std::shared_ptr<SymbolRepository> symbol_repository;
};

/**
 * @brief Tests basic functionality of the BidirectionalUnificationVisitor for a single pair of unifiable
 * literals with one applicable Function / Variable substitution.
 * @details
 *  <ul>
 *      <li>LHS Input: @f$ P \left( C\left(\right), x \right) @f$</li>
 *      <li>RHS Input: @f$ P \left( C\left(\right), D\left(\right) \right) @f$</li>
 *      <li>Expected substitutions: @f$ \left\{ x \mapsto D\left(\right) \right\} @f$</li>
 *  </ul>
 * @memberof BidirectionalUnificationTest
 */
TEST_F(BidirectionalUnificationTest, Positive_SingleBinding_FuncVar)
{
    const auto c = register_symbol<Function>("C");
    const auto d = register_symbol<Function>("D");
    const auto x = register_symbol<Variable>("x");

    const auto p1 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, x});
    const auto p2 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, d});

    EXPECT_TRUE(p1->accept(*unification_visitor, *p2));

    const Unifier expected_subs{{{x, d}}};
    EXPECT_EQ(*unification_visitor->observe_substitutions(), expected_subs);
}

/**
 * @brief Tests basic functionality of the BidirectionalUnificationVisitor for a single pair of unifiable
 * literals with two applicable Function / Variable substitutions.
 * @details
 *  <ul>
 *      <li>LHS Input: @f$ P \left( C\left(\right), x \right) @f$</li>
 *      <li>RHS Input: @f$ P \left( y, D\left(\right) \right) @f$</li>
 *      <li>Expected substitutions: @f$ \left\{ x \mapsto D\left(\right), y \mapsto C\left(\right) \right\}
 * @f$</li>
 *  </ul>
 * @memberof BidirectionalUnificationTest
 */
TEST_F(BidirectionalUnificationTest, Positive_MultipleBindings_FuncVar)
{
    const auto c = register_symbol<Function>("C");
    const auto d = register_symbol<Function>("D");
    const auto x = register_symbol<Variable>("x");
    const auto y = register_symbol<Variable>("y");

    const auto p1 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, x});
    const auto p2 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{y, d});

    EXPECT_TRUE(p1->accept(*unification_visitor, *p2));

    const Unifier expected_subs{{{x, d}, {y, c}}};
    EXPECT_EQ(*unification_visitor->observe_substitutions(), expected_subs);
```

```
}

/**
 * @brief Tests basic functionality of the BidirectionalUnificationVisitor for a single pair of unifiable
 * literals with two applicable Variable / Variable substitutions.
 * @details
 *  <ul>
 *      <li>LHS Input: @f$ P \left( a, b \right) @f$</li>
 *      <li>RHS Input: @f$ P \left( c, d \right) @f$</li>
 *      <li>Expected substitutions: @f$ \left\{ c \mapsto a, d \mapsto b \right\} @f$</li>
 *  </ul>
 * @memberof BidirectionalUnificationTest
 */
TEST_F(BidirectionalUnificationTest, Positive_MultipleBindings_VarVar)
{
    const auto a = register_symbol<Variable>("a");
    const auto b = register_symbol<Variable>("b");
    const auto c = register_symbol<Variable>("c");
    const auto d = register_symbol<Variable>("d");

    const auto p1 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{a, b});
    const auto p2 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, d});

    EXPECT_TRUE(p1->accept(*unification_visitor, *p2));

    const Unifier expected_subs{{{a, c}, {b, d}}};
    EXPECT_EQ(*unification_visitor->observe_substitutions(), expected_subs);
}

/**
 * @brief Tests basic functionality of the BidirectionalUnificationVisitor for a single pair of non-unifiable
 * literals.
 * @details
 *  <ul>
 *      <li>LHS Input: @f$ P \left( C\left(\right), x \right) @f$</li>
 *      <li>RHS Input: @f$ P \left( x, D\left(\right) \right) @f$</li>
 *  </ul>
 * @memberof BidirectionalUnificationTest
 */
TEST_F(BidirectionalUnificationTest, Negative_MultipleBindings_VarVar)
{
    const auto x = register_symbol<Variable>("x");
    const auto c = register_symbol<Function>("C");
    const auto d = register_symbol<Function>("D");

    const auto p1 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, x});
    const auto p2 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{x, d});

    EXPECT_FALSE(p1->accept(*unification_visitor, *p2));
}

/**
 * @brief Tests occurs-checking functionality of the BidirectionalUnificationVisitor for a single pair of
 * literals, non-unifiable due to the trivial case of the occurs-check (does not require substitution to
 * discover).
 * @details
 *  <ul>
 *      <li>LHS Input: @f$ P \left( C\left(\right), x \right) @f$</li>
 *      <li>RHS Input: @f$ P \left( C\left(\right), F\left(x\right) \right) @f$</li>
 *  </ul>
 * @memberof BidirectionalUnificationTest
 */
TEST_F(BidirectionalUnificationTest, Negative_OccursCheck_Trivial)
{
    const auto x = register_symbol<Variable>("x");
    const auto c = register_symbol<Function>("C");
    const auto f = register_symbol<Function>("F", std::vector<const IProcessedTerm *>{x});

    const auto p1 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, x});
    const auto p2 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, f});

    EXPECT_FALSE(p1->accept(*unification_visitor, *p2));
}

/**
 * @brief Tests occurs-checking functionality of the BidirectionalUnificationVisitor for a single pair of
 * literals, non-unifiable due to the non-trivial case of the occurs-check (requires substitution to
 * discover).
 * @details
 *  <ul>
 *      <li>LHS Input: @f$ P \left( x, F\left(x\right) \right) @f$</li>
```

```
 *      <li>RHS Input: @f$ P \left( G\left(y\right), y \right) @f$</li>
 *  </ul>
 *  <p>
 *      The substitution @f$ \left[ x \mapsto G\left(y\right) \right] @f$ is discovered, followed by the
 * substitution
 *      @f$ \left[ y \mapsto F\left(x\right) \right] @f$. Although @f$ y @f$ does not immediately appear in
 * its proposed binding @f$ F\left(x\right) @f$, expanding on the first substitution to produce
 *      @f$ F\left(G\left( y \right)\right) @f$ exposes the cycle that would be introduced by adding the
 * second substitution. Hence the LHS and RHS predicates are non-unifiable.
 *  </p>
 * @memberof BidirectionalUnificationTest
 */
TEST_F(BidirectionalUnificationTest, Negative_OccursCheck_Substituted)
{
    const auto x = register_symbol<Variable>("x");
    const auto f_x = register_symbol<Function>("F", std::vector<const IProcessedTerm *>{x});
    const auto y = register_symbol<Variable>("y");
    const auto g_y = register_symbol<Function>("G", std::vector<const IProcessedTerm *>{y});

    const auto p1 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{x, f_x});
    const auto p2 = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{g_y, y});

    EXPECT_FALSE(p1->accept(*unification_visitor, *p2));
}

} // namespace optifol
```

1.14.2 CNFNormalisationTest.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Test normalisation of FOL sentences into Conjunctive Normal Form
 * @author Oliver Dixon
 * @date 2025-06-08
 * @version Development
 */

#include <gtest/gtest.h>

#include "../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../IR/MutableVariants/Sentences/MutablePredicate.hpp"
#include "../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/DisjunctionDistributionVisitor.hpp"
#include "../Visitors/MutableTargets/Sentences/CNFNormalisers/ImplicationEliminationVisitor.hpp"
#include "GoogleTestSupport.hpp"

namespace optifol
{

/**
 * @class CNFNormalisationTest
 * @brief The CNFNormalisationTest Google Test fixture contains tests of the CNF normalisation visitor-based
 * pipeline.
 * @details <p>The seven-stage pipeline is tested:</p>
 *  <ol>
 *      <li>ImplicationEliminationVisitor</li>
 *      <li>DMLVisitor</li>
 *      <li>SymbolStandardisingVisitor</li>
 *      <li>QuantifierExtractingVisitor</li>
 *      <li>SkolemIntroducingVisitor</li>
 *      <li>UniversalEliminationVisitor</li>
 *      <li>DisjunctionDistributionVisitor</li>
 *  </ol>
 */
class CNFNormalisationTest : public testing::Test
{
protected:
    /**
     * @brief Despatches the templated CNF visitor on the given test node and verifies that the CNF-normalised
     * result matches the expected sentence construction.
     * @tparam CNFVisitor The CNF Visitor type to instantiate and despatch on the sentence
     * @param test The sentence on which the CNF Visitor should be tested
     * @param expected The expected sentence following transformation by the CNF Visitor
     * @note It is not necessary nor forbidden to wrap nodes in any level of MutableSentenceRoot objects.
```

```
     * @note The equality functor is hash-based; in particular, a commutative hash-combining function is used
     * for binary-operand sentences. Thus the expected sentence need not pass operands to commutative
     * operators in the same order as would be produced by the CNF Visitor.
     */
    template<typename CNFVisitor>
        requires std::derived_from<CNFVisitor, MutatingSentenceVisitorBase>
    static void cnf_test(
            std::unique_ptr<IMutableSentence> &&test, std::unique_ptr<IMutableSentence> &&expected)
    {
        CNFVisitor visitor;
        test->accept(visitor);
        GoogleTestSupport::test_sentence_equality(*test, *expected);
    }
};

/**
 * @brief Tests basic non-nested functionality of the ImplicationEliminationVisitor for single-operand
 * implications.
 * @details
 *  <ul>
 *      <li>Input: @f$ P \implies Q @f$</li>
 *      <li>Expected output: @f$ \lnot P \lor Q @f$</li>
 *  </ul>
 * @memberof CNFNormalisationTest
 */
TEST_F(CNFNormalisationTest, ImplicationElimination_Basic)
{
    // clang-format off
    cnf_test<ImplicationEliminationVisitor>(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Implication,
            MutablePredicate::build("P"),
            MutablePredicate::build("Q")
        ),

        MutableBinaryConnected::build(
            BinaryOperatorTypes::Disjunction,
            MutablePredicate::build("P", false),
            MutablePredicate::build("Q")
        )
    );
    // clang-format on
}

/**
 * @brief Tests basic non-nested functionality of the ImplicationEliminationVisitor for dual-operand
 * implications.
 * @details
 *  <ul>
 *      <li>Input: @f$ P \iff Q @f$</li>
 *      <li>Expected output: @f$ \left( P \lor \lnot Q \right) \land \left( \lnot P \lor Q \right) @f$</li>
 *  </ul>
 * @memberof CNFNormalisationTest
 */
TEST_F(CNFNormalisationTest, ImplicationElimination_Biconditional)
{
    // clang-format off
    cnf_test<ImplicationEliminationVisitor>(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Biconditional,
            MutablePredicate::build("P"),
            MutablePredicate::build("Q")
        ),

        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutablePredicate::build("P"),
                MutablePredicate::build("Q", false)
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutablePredicate::build("P", false),
                MutablePredicate::build("Q")
            )
        )
    );
    // clang-format on
}
```

```
/**
 * @brief Tests functionality of the ImplicationEliminationVisitor for nesting on a single operand.
 * @details
 *  <ul>
 *      <li>Input: @f$ \left( P \implies Q \right) \iff R @f$</li>
 *      <li>Expected output: @f$ \left( \lnot \left( \lnot P \lor Q \right) \lor R \right) \land
 *          \left( \left( \lnot P \lor Q \right) \lor \lnot R \right) @f$</li>
 *  </ul>
 * @memberof CNFNormalisationTest
 */
TEST_F(CNFNormalisationTest, ImplicationElimination_UnaryNesting)
{
    // clang-format off
    cnf_test<ImplicationEliminationVisitor>(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Biconditional,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Implication,
                MutablePredicate::build("P"),
                MutablePredicate::build("Q")
            ),
            MutablePredicate::build("R")
        ),

        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("P", false),
                    MutablePredicate::build("Q"),
                    false
                ),
                MutablePredicate::build("R")
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("P", false),
                    MutablePredicate::build("Q")
                ),
                MutablePredicate::build("R", false)
            )
        )
    );
    // clang-format on
}

/**
 * @brief Tests functionality of the ImplicationEliminationVisitor for nesting on both operands.
 * @details
 *  <ul>
 *      <li>Input: @f$ \left( P \implies Q \right) \iff \left( R \implies S \right) @f$</li>
 *      <li>
 *          Expected output:
 *          @f$ \left( \lnot \left( \lnot P \lor Q \right) \lor \left( \lnot R \lor S \right) \right) \land
 *          \left( \left( \lnot P \lor Q \right) \lor \lnot \left( \lnot R \lor S \right) \right)@f$
 *      </li>
 *  </ul>
 * @memberof CNFNormalisationTest
 */
TEST_F(CNFNormalisationTest, ImplicationElimination_BinaryNesting)
{
    // clang-format off
    cnf_test<ImplicationEliminationVisitor>(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Biconditional,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Implication,
                MutablePredicate::build("P"),
                MutablePredicate::build("Q")
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Implication,
                MutablePredicate::build("R"),
                MutablePredicate::build("S")
            )
        ),
```

```
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("P", false),
                    MutablePredicate::build("Q"),
                    false
                ),
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("R", false),
                    MutablePredicate::build("S")
                )
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("P", false),
                    MutablePredicate::build("Q")
                ),
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("R", false),
                    MutablePredicate::build("S"),
                    false
                )
            )
        )
    );
    // clang-format on
}

/**
 * @brief Tests functionality of the DisjunctionDistributionVisitor for distribution over a single operand.
 * @details
 *  <ul>
 *      <li>Input: @f$ P \lor \left( Q \land R \right) @f$</li>
 *      <li>Expected output: @f$ \left( P \lor Q \right) \land \left( P \lor R \right) @f$</li>
 *  </ul>
 * @memberof CNFNormalisationTest
 */
TEST_F(CNFNormalisationTest, DisjunctionDistribution_BasicUnary)
{
    // clang-format off
    cnf_test<DisjunctionDistributionVisitor>(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Disjunction,
            MutablePredicate::build("P"),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Conjunction,
                MutablePredicate::build("Q"),
                MutablePredicate::build("R")
            )
        ),

        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutablePredicate::build("P"),
                MutablePredicate::build("Q")
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutablePredicate::build("P"),
                MutablePredicate::build("R")
            )
        )
    );
    // clang-format on
}

/**
 * @brief Tests functionality of the DisjunctionDistributionVisitor for distribution over both operands.
 * @details
 *  <ul>
 *      <li>Input: @f$ \left( P \lor \left( Q \land R \right) \right) \land
 *          \left( \left( A \land B \right) \lor C \right) @f$</li>
```

```
 *      <li>
 *          Expected output:
 *          @f$ \left( \left( P \lor Q \right) \land \left( P \lor R \right) \right) \land
 *          \left( \left( C \lor A \right) \land \left( C \lor B \right) \right) @f$
 *      </li>
 *  </ul>
 * @memberof CNFNormalisationTest
 */
TEST_F(CNFNormalisationTest, DisjunctionDistribution_BasicBinary)
{
    // clang-format off
    cnf_test<DisjunctionDistributionVisitor>(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutablePredicate::build("P"),
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Conjunction,
                    MutablePredicate::build("Q"),
                    MutablePredicate::build("R")
                )
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Conjunction,
                    MutablePredicate::build("A"),
                    MutablePredicate::build("B")
                ),
                MutablePredicate::build("C")
            )
        ),

        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Conjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("P"),
                    MutablePredicate::build("Q")
                ),
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("P"),
                    MutablePredicate::build("R")
                )
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Conjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("C"),
                    MutablePredicate::build("A")
                ),
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("C"),
                    MutablePredicate::build("B")
                )
            )
        )
    );
    // clang-format on
}

/**
 * @brief Tests functionality of the DisjunctionDistributionVisitor for distribution over both operands, where
 * a reduction is only applicable to one.
 * @details
 *  <ul>
 *      <li>Input: @f$ A \land \left( P \lor \left( Q \land R \right) \right) @f$</li>
 *      <li>Expected output: @f$ A \land \left( \left( P \lor Q \right) \land \left( P \lor R \right) \right)
 * @f$</li>
 *  </ul>
 * @memberof CNFNormalisationTest
 */
TEST_F(CNFNormalisationTest, DisjunctionDistribution_NestedNoOp)
{
    // clang-format off
```

```
    cnf_test<DisjunctionDistributionVisitor>(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutablePredicate::build("A"),
            MutableBinaryConnected::build(
            BinaryOperatorTypes::Disjunction,
            MutablePredicate::build("P"),
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Conjunction,
                    MutablePredicate::build("Q"),
                    MutablePredicate::build("R")
                )
            )
        ),

        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutablePredicate::build("A"),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Conjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("P"),
                    MutablePredicate::build("Q")
                ),
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("P"),
                    MutablePredicate::build("R")
                )
            )
        )
    );
    // clang-format on
}

} // namespace optifol
```

1.14.3 FeatureVectorIndexTest.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Test the Feature Vector Indexing on Clauses
 * @author Oliver Dixon
 * @date 2026-01-30
 * @version Development
 */

#include <gtest/gtest.h>

#include "../IR/Sentences/Clause.hpp"
#include "../IR/Sentences/Literal.hpp"
#include "../IR/SymbolRepository.hpp"
#include "../IR/Terms/Function.hpp"
#include "../Inference/FVIKnowledgeBase.hpp"

namespace optifol
{

class FeatureVectorIndexTest : public testing::Test
{
protected:
    void SetUp() override
    {
        symbol_repository = std::make_shared<SymbolRepository>();
    }

    std::shared_ptr<SymbolRepository> symbol_repository;
};

/**
 * @brief
 * @details
 * @memberof FeatureVectorIndexTest
```

```
 */
TEST_F(FeatureVectorIndexTest, Test1)
{
    const auto p_literal = symbol_repository->add_symbol(Literal::build("P"));
    const auto q_literal = symbol_repository->add_symbol(Literal::build("Q"));
    const auto r_literal = symbol_repository->add_symbol(Literal::build("R"));

    auto c1 = Clause::build();
    c1->add_literal(p_literal);
    c1->add_literal(q_literal);

    auto c2 = Clause::build();
    c2->add_literal(p_literal);
    c2->add_literal(r_literal);

    FVIKnowledgeBase kb(symbol_repository);
    kb.add_clause(std::move(c1));
    kb.add_clause(std::move(c2));

    auto ct = Clause::build();
    ct->add_literal(p_literal);

    kb.add_clause(std::move(ct));
}

} // namespace optifol
```

1.14.4 FOLParserTest.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Test lexing and parsing of FOL sentences in the Optifol grammar
 * @author Oliver Dixon
 * @date 2025-02-16
 * @version Development
 */

#include <gtest/gtest.h>

#include "../Interpreter/FOLLexer.hpp"
#include "GoogleTestSupport.hpp"

namespace optifol
{

/**
 * @class FOLParserTest
 * @brief Provide a convenient input-streamer to the FOL lexer and parser for use with the Google Test
 * framework
 */
class FOLParserTest : public testing::Test
{
protected:
    void TearDown() override
    {
        lexer_input_stream.clear();
    }

    /**
     * @brief Tests that the given input correctly lexes and parses to the equivalent given typed structure.
     * @param test The raw input string to pass to the lexer.
     * @param expected The expected output sentence structure. The sentence is wrapped in a positive
     *  MutableSentenceRoot.
     */
    void equality_on_input(const char *test, std::unique_ptr<IMutableSentence> &&expected)
    {
        lexer_input_stream.str(test);
        parser.parse();

        const auto sentence = parser.retrieve_sentence();
        const auto expected_root = std::make_unique<MutableSentenceRoot>(std::move(expected));

        GoogleTestSupport::test_sentence_equality(*sentence, *expected_root);
    }
```

```
private:
    std::istringstream lexer_input_stream;
    FOLLexer lexer{lexer_input_stream, std::cerr};
    FOLParser parser{&lexer};
};

TEST_F(FOLParserTest, Quantifier_Universal)
{
    std::vector<std::unique_ptr<IMutableTerm>> p_args;
    p_args.push_back(MutableVariable::build<IMutableTerm>("x"));

    // clang-format off
    equality_on_input(
        "%Ux(P(x))",

        MutableQuantified::build(
            QuantifierTypes::Universal,
            MutableVariable::build("x"),
            MutablePredicate::build("P", std::move(p_args))
        )
    );
}

TEST_F(FOLParserTest, Quantifier_NegativeExistential)
{
    std::vector<std::unique_ptr<IMutableTerm>> q_args;
    q_args.push_back(MutableVariable::build<IMutableTerm>("y"));

    // clang-format off
    equality_on_input(
        "%Ey(~Q(y))",

        MutableQuantified::build(
            QuantifierTypes::Existential,
            MutableVariable::build("y"),
            MutablePredicate::build("Q", false, std::move(q_args))
        )
    );
}

TEST_F(FOLParserTest, Quantifier_Nested)
{
    std::vector<std::unique_ptr<IMutableTerm>> p_args;
    p_args.push_back(MutableVariable::build<IMutableTerm>("x"));

    std::vector<std::unique_ptr<IMutableTerm>> q_args;
    q_args.push_back(MutableVariable::build<IMutableTerm>("y"));

    // clang-format off
    equality_on_input(
        "%Ux(%Ey(P(x) & Q(y)))",

        MutableQuantified::build(
            QuantifierTypes::Universal,
            MutableVariable::build("x"),
            MutableQuantified::build(
                QuantifierTypes::Existential,
                MutableVariable::build("y"),
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Conjunction,
                    MutablePredicate::build("P", std::move(p_args)),
                    MutablePredicate::build("Q", std::move(q_args))
                )
            )
        )
    );
}

TEST_F(FOLParserTest, TermBuilder_NoArguments)
{
    // clang-format off
    equality_on_input(
        "%Ux(P())",

        MutableQuantified::build(
            QuantifierTypes::Universal,
            MutableVariable::build("x"),
            MutablePredicate::build("P", true)
        )
    );
```

```
}

}
```

1.14.5 GoogleTestSupport.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class definition and implementation of static Google Test support
 * @author Oliver Dixon
 * @date 2025-06-14
 * @version Development
 */

#ifndef GOOGLETESTSUPPORT_HPP
#define GOOGLETESTSUPPORT_HPP

namespace optifol
{

// Current Clang 18 bug reports Doxygen violations for uses of @tparam on templated concepts.
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wdocumentation"
/**
 * @concept GoogleTestable
 * @brief Represents a type that has good support for use in unit tests orchestrated by the Google Test
 * framework.
 * @details In particular, a type <code>Candidate</code> with good support implements:
 *  <ul>
 *      <li>Equality comparison with <code>Candidate::operator==(const Candidate&)</code>;</li>
 *      <li>Inverse equality comparison with <code>Candidate::operator!=(const Candidate&)</code>, noting that
 * this can often be generated by compilers given a valid equality operator overload;</li> <li>Stream-bound
 * serialisation with <code>operator<<(std::ostream&, const Candidate&)</code>.</li>
 *  </ul>
 * @tparam Candidate The candidate for the concept properties
 */
template<typename Candidate>
concept GoogleTestable = requires(const Candidate &lhs, const Candidate &rhs, std::ostream &ostream) {
    { lhs.operator==(rhs) } -> std::convertible_to<bool>; // For EXPECT_EQ, et al.
    {
        ostream << lhs
    } -> std::same_as<std::ostream &>; // For serialising 'actual' vs. 'expected' results on failure.
};
#pragma clang diagnostic pop

/**
 * @class GoogleTestSupport
 * @brief Provides static member functions to support Google Test assertions for Optifol types.
 */
class GoogleTestSupport
{
public:
    /**
     * @brief Assert equality on two GoogleTestable results
     * @tparam ActualType The type of the actual returned value
     * @tparam ExpectedType The type of the expected value
     * @param actual The actual returned value
     * @param expected The expected value
     */
    template<GoogleTestable ActualType, GoogleTestable ExpectedType>
    static void test_sentence_equality(const ActualType &actual, const ExpectedType &expected)
    {
        EXPECT_EQ(actual, expected);
    }
};

} // namespace optifol

#endif
```

1.14.6 LiteralOrganisationTest.cpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Test organisation of Literal objects into Clause nodes and SentenceRoot nodes
 * @author Oliver Dixon
 * @date 2026-01-25
 * @version Development
 */

#include <algorithm>
#include <gtest/gtest.h>
#include <random>
#include <ranges>

#include "../IR/Sentences/Clause.hpp"
#include "../IR/Sentences/SentenceRoot.hpp"

namespace optifol
{

/**
 * @class LiteralOrganisationTest
 * @brief Verifies API characteristics of Literal, Clause, and SentenceRoot structures relating to reductions
 * for tautologies and bottoms, and handles duplicates.
 */
class LiteralOrganisationTest : public testing::Test
{
protected:
    /**
     * @brief Randomly generate a fixed number of non-complementary Literal objects, owned by the returned
     * container, and inserted into the given Clause.
     * @param clause_size The number of Literal objects to generate.
     * @param target The Clause into which the Literal objects should be registered.
     * @return The owning containers of the constructed Literal objects.
     */
    static std::vector<std::unique_ptr<Literal>> generate_literals(
            const unsigned int clause_size, Clause &target)
    {
        // Build the literals and hold ownership.
        std::vector<std::unique_ptr<Literal>> owning_container(clause_size);
        unsigned int literal_idx = 0;
        std::ranges::generate(
                owning_container, [&literal_idx] { return Literal::build(std::to_string(++literal_idx)); });

        // Provide weak references (through a C++23 projection) for each owning container to the given clause.
        std::ranges::for_each(
                owning_container, [&target](const Literal *const literal) { target.add_literal(literal); },
                [](const std::unique_ptr<Literal> &literal) { return literal.get(); });

        return owning_container;
    }

    /**
     * @brief Generate a fixed number of non-equal Clause objects, owned by the returned container, and
     * inserted into the given SentenceRoot.
     * @param clause_count The number of Clauses to insert into the SentenceRoot.
     * @param target The SentenceRoot into which the Clause objects should be registered.
     * @return The owning containers of the constructed Literal objects.
     */
    static std::vector<std::unique_ptr<Literal>> generate_clauses(
            const unsigned int clause_count, SentenceRoot &target)
    {
        std::vector<std::unique_ptr<Literal>> literals_owner;

        // Set up a generator to produce clauses with increasing numbers of literals.
        auto clauses_view = std::views::iota(0U) |
                std::views::transform(
                        [&literals_owner](const auto idx) -> Clause
                        {
                            Clause clause;
                            auto new_literals_owner = generate_literals(idx, clause);
                            literals_owner.insert(literals_owner.end(),
                                    std::make_move_iterator(new_literals_owner.begin()),
                                    std::make_move_iterator(new_literals_owner.end()));
                            return clause;
                        });
```

```
        // Sample the fixed number of clauses for the SentenceRoot.
        for (auto clause: clauses_view | std::views::take(clause_count))
            target.add_clause(std::move(clause));

        return literals_owner;
    }
};

/**
 * @brief Verify that a Clause can accept a single pair of complementary Literal objects and reduce to a
 * tautology.
 * @memberof LiteralOrganisationTest
 */
TEST_F(LiteralOrganisationTest, ClauseTriviallyTrue_Simple)
{
    const auto p = Literal::build("P");
    const auto p_complement = Literal::build("P", std::initializer_list<const IProcessedTerm *>{}, false);

    const Clause clause{p.get(), p_complement.get()};

    EXPECT_EQ(clause.get_triviality_state(), Clause::State::TriviallyTrue);
    EXPECT_EQ(clause.order(), 0U);
}

/**
 * @brief Verify that a Clause can accept a large number of Literal objects, in any order, and detect
 * tautologies.
 * @memberof LiteralOrganisationTest
 */
TEST_F(LiteralOrganisationTest, ClauseTriviallyTrue_Complex)
{
    static constexpr unsigned int clause_size = 64;
    Clause clause;
    auto literals = generate_literals(clause_size, clause);

    /*
     * Randomly generate and add the complement of one existing literal, and verify that it pulls the clause
     * into a tautology.
     */
    std::random_device random_device;
    std::mt19937 rng(random_device());
    std::uniform_int_distribution<> dist(1, clause_size);

    literals.emplace_back(Literal::build(
            std::to_string(dist(rng)), std::initializer_list<const IProcessedTerm *>{}, false));
    clause.add_literal(literals.back().get());

    EXPECT_EQ(clause.get_triviality_state(), Clause::State::TriviallyTrue);
    EXPECT_EQ(clause.order(), 0U);
}

/**
 * @brief Verify that an empty Clause is correctly marked as trivially false.
 * @memberof LiteralOrganisationTest
 */
TEST_F(LiteralOrganisationTest, ClauseTriviallyFalse)
{
    const Clause clause{};

    EXPECT_EQ(clause.get_triviality_state(), Clause::State::TriviallyFalse);
    EXPECT_EQ(clause.order(), 0U);
}

/**
 * @brief Verify that a Clause with no complementary pairs is correctly marked as non-trivial and accepts all
 * Literal objects.
 * @memberof LiteralOrganisationTest
 */
TEST_F(LiteralOrganisationTest, ClauseNonTrivial_Simple)
{
    static constexpr unsigned int clause_size = 32;
    Clause clause;

    [[maybe_unused]] auto literals = generate_literals(clause_size, clause);

    EXPECT_EQ(clause.get_triviality_state(), Clause::State::NotTrivial);
    EXPECT_EQ(clause.order(), clause_size);
}

/**
 * @brief Verify that a Clause with no complementary pairs and duplicates is correctly marked as non-trivial
```

```
 * and denies only non-unique Literal objects.
 * @memberof LiteralOrganisationTest
 */
TEST_F(LiteralOrganisationTest, ClauseNonTrivial_Duplicates)
{
    static constexpr unsigned int group_size = 8;
    Clause clause;

    [[maybe_unused]] auto literals_group_1 = generate_literals(group_size, clause);
    EXPECT_EQ(clause.order(), group_size);

    [[maybe_unused]] auto literals_group_2 = generate_literals(group_size, clause);

    // Crucially, check that the literal duplicates did not increase the order of the clause.
    EXPECT_EQ(clause.order(), group_size);
}

/**
 * @brief Verify that a SentenceRoot with distinct Clause objects accepts them all.
 * @memberof LiteralOrganisationTest
 */
TEST_F(LiteralOrganisationTest, SentenceRoot_Simple)
{
    static constexpr unsigned int clause_count = 8;
    SentenceRoot root;

    [[maybe_unused]] auto literals_owner = generate_clauses(clause_count, root);

    EXPECT_TRUE(root.order() == clause_count);
}

/**
 * @brief Verify that a SentenceRoot refuses duplicate Clause objects.
 * @memberof LiteralOrganisationTest
 */
TEST_F(LiteralOrganisationTest, SentenceRoot_Duplicates)
{
    static constexpr unsigned int group_size = 8;
    SentenceRoot root;

    [[maybe_unused]] auto literals_group_1 = generate_clauses(group_size, root);
    EXPECT_EQ(root.order(), group_size);

    [[maybe_unused]] auto literals_group_2 = generate_clauses(group_size, root);

    // Crucially, check that the clause duplicates did not increase the order of the SentenceRoot.
    EXPECT_EQ(root.order(), group_size);
}

} // namespace optifol
```

1.14.7 RepositoryBuildingTest.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Test registration of CNF-normalised FOL sentences into a centralised SymbolRepository
 * @author Oliver Dixon
 * @date 2025-09-10
 * @version Development
 */

#include <gtest/gtest.h>

#include "../IR/MutableVariants/Sentences/IMutableSentence.hpp"
#include "../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../IR/MutableVariants/Sentences/MutablePredicate.hpp"
#include "../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"
#include "../IR/MutableVariants/Terms/MutableVariable.hpp"
#include "../IR/Sentences/Literal.hpp"
#include "../IR/SymbolRepository.hpp"
#include "../IR/Terms/Variable.hpp"
#include "../Visitors/MutableTargets/RepositoryBuildingVisitor.hpp"

namespace optifol
```

```
{

/**
 * @class RepositoryBuildingTest
 * @brief The RepositoryBuildingTest Google Test fixture contains tests of the RepositoryBuildingVisitor.
 */
class RepositoryBuildingTest : public testing::Test
{
protected:
    /**
     * @brief Despatches the RepositoryBuildingVisitor with a fresh SymbolRepository over the given
     * IMutableSentence and compares the actual results to the expected.
     * @param test The initial node to test, not wrapped inside of a MutableSentenceRoot. The tree must be
     * normalised to CNF.
     * @param expected The expected set of ordered clauses, each of which contains the expected set of ordered
     * literals.
     */
    static void repo_build_test(std::unique_ptr<IMutableSentence> &&test,
            const std::vector<std::vector<const Literal *>> &expected)
    {
        const auto repository = std::make_shared<SymbolRepository>();
        RepositoryBuildingVisitor visitor(repository);
        const auto wrapped_test_input = std::make_unique<MutableSentenceRoot>(std::move(test));

        wrapped_test_input->accept(visitor);
        const auto transformed_root = visitor.take_last_root();

        auto actual_clause_begin = transformed_root->begin();
        const auto actual_clause_end = transformed_root->end();

        for (const auto &expected_clause: expected) {
            // Verify that the actual clause list does not exceed the expected clause list.
            EXPECT_NE(actual_clause_begin, actual_clause_end);

            auto actual_literal_begin = actual_clause_begin->begin();
            const auto actual_literal_end = actual_clause_begin->end();

            for (const auto expected_literal: expected_clause) {
                // Verify that the actual literal list does not exceed the expected literal list for the fixed
                // clause.
                EXPECT_NE(actual_literal_begin, actual_literal_end);

                // Verify equality of the Literals.
                EXPECT_EQ(**actual_literal_begin, *expected_literal);

                // Verify insertion into repository.
                EXPECT_NE(repository->get_symbol_handle(*expected_literal), nullptr);

                ++actual_literal_begin;
            }

            // Verify that the actual literal list is exhausted at the same point as the expected literal
            // list.
            EXPECT_EQ(actual_literal_begin, actual_literal_end);
            ++actual_clause_begin;
        }

        // Verify that the actual clause list is exhausted at the same point as the expected clause list.
        EXPECT_EQ(actual_clause_begin, actual_clause_end);
    }
};

/**
 * @brief Tests basic functionality of the RepositoryBuildingVisitor for a single predicate with arguments.
 * @details
 *  <ul>
 *      <li>Input: @f$ P \left( x, y, z \right) @f$</li>
 *      <li>Expected output: @f$ \left\{ \left\{ P \left( x_0, y_0, z_0 \right) \right\} \right\} @f$, where
 * the subscripted suffixes are introduced to disambiguate across sentences.</li>
 *  </ul>
 * @memberof RepositoryBuildingTest
 */
TEST_F(RepositoryBuildingTest, SingleClause_SinglePredicate)
{
    // clang-format off

    std::vector<std::unique_ptr<IMutableTerm>> p_args;
    p_args.push_back(MutableVariable::build<IMutableTerm>("x"));
    p_args.push_back(MutableVariable::build<IMutableTerm>("y"));
    p_args.push_back(MutableVariable::build<IMutableTerm>("z"));
```

```
    const Variable x("x", "x#0");
    const Variable y("y", "y#0");
    const Variable z("z", "z#0");
    const Literal p("P", { &x, &y, &z });

    repo_build_test(
        MutablePredicate::build("P", std::move(p_args)),
        { { &p } }
    );

    // clang-format on
}

/**
 * @brief Tests basic functionality of the RepositoryBuildingVisitor for multiple predicates within a single
 * CNF clause.
 * @details
 *  <ul>
 *      <li>Input: @f$ P \lor Q @f$</li>
 *      <li>Expected output: @f$ \left\{ \left\{ P, Q \right\} \right\} @f$</li>
 *  </ul>
 * @memberof RepositoryBuildingTest
 */
TEST_F(RepositoryBuildingTest, SingleClause_MultiplePredicates)
{
    // clang-format off

    const Literal p("P");
    const Literal q("Q");

    repo_build_test(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Disjunction,
            MutablePredicate::build("P"),
            MutablePredicate::build("Q")
        ),

        { { &p, &q } }
    );

    // clang-format on
}

/**
 * @brief Tests basic functionality of the RepositoryBuildingVisitor for single predicates over multiple CNF
 * clauses.
 * @details
 *  <ul>
 *      <li>Input: @f$ P \land Q @f$</li>
 *      <li>Expected output: @f$ \left\{ \left\{ P \right\}, \left\{ Q \right\} \right\} @f$</li>
 *  </ul>
 * @memberof RepositoryBuildingTest
 */
TEST_F(RepositoryBuildingTest, MultipleClauses_SinglePredicate)
{
    // clang-format off

    const Literal p("P");
    const Literal q("Q");

    repo_build_test(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutablePredicate::build("P"),
            MutablePredicate::build("Q")
        ),

        { { &p }, { &q } }
    );

    // clang-format on
}

/**
 * @brief Tests basic functionality of the RepositoryBuildingVisitor for multiple predicates over multiple CNF
 * clauses.
 * @details
 *  <ul>
 *      <li>Input: @f$ \left( P \lor Q \right) \land \left( R \lor S \right) @f$</li>
 *      <li>Expected output: @f$ \left\{ \left\{ P, Q \right\}, \left\{ R, S \right\} \right\} @f$</li>
 *  </ul>
```

```
 * @memberof RepositoryBuildingTest
 */
TEST_F(RepositoryBuildingTest, MultipleClauses_MultiplePredicates)
{
    // clang-format off

    const Literal p("P");
    const Literal q("Q");
    const Literal r("R");
    const Literal s("S");

    repo_build_test(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutablePredicate::build("P"),
                MutablePredicate::build("Q")
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutablePredicate::build("R"),
                MutablePredicate::build("S")
            )
        ),

        { { &p, &q }, { &r, &s } }
    );

    // clang-format on
}

/**
 * @brief Tests basic functionality of the RepositoryBuildingVisitor for multiple predicates over multiple CNF
 * clauses, including nested CNF formulas.
 * @details
 *  <ul>
 *      <li>
 *          Input:
 *          @f$ \left( \left( A \lor B \right) \lor C \right) \land
 *          \left( D \land \left( E \land \left( F \lor G \right) \right) \right) @f$
 *      </li>
 *      <li>
 *          Expected output:
 *          @f$\left\{
 *              \left\{ A, B, C \right\},
 *              \left\{ D \right\},
 *              \left\{ E \right\},
 *              \left\{ F, G \right\}
 *          \right\}@f$
 *      </li>
 *  </ul>
 * @memberof RepositoryBuildingTest
 */
TEST_F(RepositoryBuildingTest, Nested)
{
    // clang-format off

    const Literal a("A");
    const Literal b("B");
    const Literal c("C");
    const Literal d("D");
    const Literal e("E");
    const Literal f("F");
    const Literal g("G");

    repo_build_test(
        MutableBinaryConnected::build(
            BinaryOperatorTypes::Conjunction,
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Disjunction,
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("A"),
                    MutablePredicate::build("B")
                ),
                MutablePredicate::build("C")
            ),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Conjunction,
                MutablePredicate::build("D"),
```

```
                MutableBinaryConnected::build(
                    BinaryOperatorTypes::Conjunction,
                    MutablePredicate::build("E"),
                    MutableBinaryConnected::build(
                        BinaryOperatorTypes::Disjunction,
                        MutablePredicate::build("F"),
                        MutablePredicate::build("G")
                    )
                )
            )
        ),

        {
            { &a, &b, &c },
            { &d },
            { &e },
            { &f, &g }
        }
    );

    // clang-format on
}

} // namespace optifol
```

1.14.8 ResolutionTest.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Test resolution of FOL knowledge bases to deduce goals
 * @author Oliver Dixon
 * @date 2025-09-13
 * @version Development
 */

#include <gtest/gtest.h>

#include "../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../IR/MutableVariants/Sentences/MutablePredicate.hpp"
#include "../IR/MutableVariants/Sentences/MutableQuantified.hpp"
#include "../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"
#include "../IR/MutableVariants/Terms/MutableFunction.hpp"
#include "../IR/MutableVariants/Terms/MutableVariable.hpp"
#include "../IR/SymbolRepository.hpp"
#include "../Inference/ExpressionFactory.hpp"
#include "../Inference/Prover.hpp"
#include "../Inference/QueryResult.hpp"

namespace optifol
{

/**
 * @class ResolutionTest
 */
class ResolutionTest : public testing::Test
{
protected:
    void SetUp() override
    {
        symbol_repository = std::make_shared<SymbolRepository>();
        prover = std::make_unique<Prover>(symbol_repository);
    }

    std::shared_ptr<SymbolRepository> symbol_repository;
    std::unique_ptr<Prover> prover;
};

/**
 * @brief Tests Resolution functionality for a Modus Ponens knowledge base over a single universally
 * quantified variable
 *  @f$ x @f$ and a constant @f$ C @f$.
 * @details
 *  <table>
 *      <tr>
```

```
 *          <td />
 *          <th>FOL Sentence</th>
 *          <th>CNF Sentence</th>
 *          <th>Conjunct-Disjunct Set</th>
 *      </tr>
 *      <tr>
 *          <th rowspan="2">Knowledge Base</th>
 *          <td>@f$ \forall x \left( P\left(x\right) \implies Q\left(x\right) \right) @f$</td>
 *          <td>@f$ \lnot P\left(x\right) \lor Q\left(x\right) @f$</td>
 *          <td>@f$ \left\{ \left\{ \lnot P\left(x\right), Q\left(x\right) \right\} \right\} @f$</td>
 *      </tr>
 *      <tr>
 *          <td>@f$ P\left(C\right) @f$</td>
 *          <td>@f$ P\left(C\right) @f$</td>
 *          <td>@f$ \left\{ \left\{ P\left(C\right) \right\} \right\} @f$</td>
 *      </tr>
 *      <tr>
 *          <th>Goal</th>
 *          <td>@f$ Q\left(C\right) @f$</td>
 *          <td>@f$ Q\left(C\right) @f$</td>
 *          <td>@f$ \left\{ \left\{ Q\left(C\right) \right\} \right\} @f$</td>
 *      </tr>
 *  </table>
 * @memberof ResolutionTest
 */
TEST_F(ResolutionTest, ModusPonens_Quantified)
{
    std::vector<std::unique_ptr<IMutableTerm>> s1_p_args;
    s1_p_args.push_back(MutableVariable::build<IMutableTerm>("x"));

    std::vector<std::unique_ptr<IMutableTerm>> s1_q_args;
    s1_q_args.push_back(MutableVariable::build<IMutableTerm>("x"));

    std::vector<std::unique_ptr<IMutableTerm>> s2_p_args;
    s2_p_args.push_back(MutableFunction::build<IMutableTerm>("C"));

    // clang-format off

    const auto sentence1 = ExpressionFactory::build_sentence(MutableSentenceRoot::build(
        MutableQuantified::build(
            QuantifierTypes::Universal,
            MutableVariable::build("x"),
            MutableBinaryConnected::build(
                BinaryOperatorTypes::Implication,
                MutablePredicate::build("P", std::move(s1_p_args)),
                MutablePredicate::build("Q", std::move(s1_q_args))
            )
        )
    ), symbol_repository);

    // clang-format on

    const auto sentence2 = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(MutablePredicate::build("P", std::move(s2_p_args))),
            symbol_repository);

    prover->tell(*sentence1);
    prover->tell(*sentence2);

    std::vector<std::unique_ptr<IMutableTerm>> query_args;
    query_args.push_back(MutableFunction::build<IMutableTerm>("C"));

    const auto result =
            prover->ask(MutableSentenceRoot::build(MutablePredicate::build("Q", std::move(query_args))));

    EXPECT_EQ(result.outcome, QueryResult::ConjectureStatus::Consistent);
}

TEST_F(ResolutionTest, Reject_Trivial)
{
    std::vector<std::unique_ptr<IMutableTerm>> s1_p_args;
    s1_p_args.push_back(MutableVariable::build<IMutableTerm>("x"));

    std::vector<std::unique_ptr<IMutableTerm>> s1_q_args;
    s1_q_args.push_back(MutableVariable::build<IMutableTerm>("x"));

    std::vector<std::unique_ptr<IMutableTerm>> s2_p_args;
    s2_p_args.push_back(MutableFunction::build<IMutableTerm>("C"));

    const auto sentence1 = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(MutablePredicate::build("P", std::move(s2_p_args))),
```

```
            symbol_repository);

    const auto sentence2 = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(MutablePredicate::build("Q", std::move(s2_p_args))),
            symbol_repository);

    prover->tell(*sentence1);
    prover->tell(*sentence2);

    std::vector<std::unique_ptr<IMutableTerm>> query_args;
    query_args.push_back(MutableFunction::build<IMutableTerm>("C"));

    const auto result =
            prover->ask(MutableSentenceRoot::build(MutablePredicate::build("R", std::move(query_args))));

    EXPECT_EQ(result.outcome, QueryResult::ConjectureStatus::Inconsistent);
}

TEST_F(ResolutionTest, CuriosityKilledTheCat)
{
    // Anybody who loves all animals is themselves loved by somebody.
    std::vector<std::unique_ptr<IMutableTerm>> animal_args_1;
    animal_args_1.push_back(MutableVariable::build("y"));

    std::vector<std::unique_ptr<IMutableTerm>> loves_args_1;
    loves_args_1.push_back(MutableVariable::build("x"));
    loves_args_1.push_back(MutableVariable::build("y"));

    std::vector<std::unique_ptr<IMutableTerm>> loves_args_2;
    loves_args_2.push_back(MutableVariable::build("y"));
    loves_args_2.push_back(MutableVariable::build("x"));

    const auto loves_all_animals = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(MutableQuantified::build(QuantifierTypes::Universal,
                    MutableVariable::build("x"),
                    MutableBinaryConnected::build(BinaryOperatorTypes::Implication,
                            MutableQuantified::build(QuantifierTypes::Universal, MutableVariable::build("y"),
                                    MutableBinaryConnected::build(BinaryOperatorTypes::Implication,
                                            MutablePredicate::build("Animal", std::move(animal_args_1)),
                                            MutablePredicate::build("Loves", std::move(loves_args_1)))),
                            MutableQuantified::build(QuantifierTypes::Existential,
                                    MutableVariable::build("y"),
                                    MutablePredicate::build("Loves", std::move(loves_args_2)))))),
            symbol_repository);

    // Anybody who kills an animal is loved by nobody.
    std::vector<std::unique_ptr<IMutableTerm>> animal_args_2;
    animal_args_2.push_back(MutableVariable::build("z"));

    std::vector<std::unique_ptr<IMutableTerm>> kills_args_1;
    kills_args_1.push_back(MutableVariable::build("x"));
    kills_args_1.push_back(MutableVariable::build("z"));

    std::vector<std::unique_ptr<IMutableTerm>> loves_args_3;
    loves_args_3.push_back(MutableVariable::build("y"));
    loves_args_3.push_back(MutableVariable::build("x"));

    const auto kills_an_animal = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(MutableQuantified::build(QuantifierTypes::Universal,
                    MutableVariable::build("x"),
                    MutableBinaryConnected::build(BinaryOperatorTypes::Implication,
                            MutableQuantified::build(QuantifierTypes::Existential,
                                    MutableVariable::build("z"),
                                    MutableBinaryConnected::build(BinaryOperatorTypes::Conjunction,
                                            MutablePredicate::build("Animal", std::move(animal_args_2)),
                                            MutablePredicate::build("Kills", std::move(kills_args_1)))),
                            MutableQuantified::build(QuantifierTypes::Universal, MutableVariable::build("y"),
                                    MutablePredicate::build("Loves", false, std::move(loves_args_3)))))),
            symbol_repository);

    // Jack loves all animals.
    std::vector<std::unique_ptr<IMutableTerm>> animal_args_3;
    animal_args_3.push_back(MutableVariable::build("x"));

    std::vector<std::unique_ptr<IMutableTerm>> loves_args_4;
    loves_args_4.push_back(MutableFunction::build("Jack"));
    loves_args_4.push_back(MutableVariable::build("x"));

    const auto jack_loves_animals = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(
                    MutableQuantified::build(QuantifierTypes::Universal, MutableVariable::build("x"),
```

```
                        MutableBinaryConnected::build(BinaryOperatorTypes::Implication,
                                MutablePredicate::build("Animal", std::move(animal_args_3)),
                                MutablePredicate::build("Loves", std::move(loves_args_4)))),
            symbol_repository);

    // Tuna is killed by Jack or Curiosity.
    std::vector<std::unique_ptr<IMutableTerm>> kills_args_2;
    kills_args_2.push_back(MutableFunction::build("Jack"));
    kills_args_2.push_back(MutableFunction::build("Tuna"));

    std::vector<std::unique_ptr<IMutableTerm>> kills_args_3;
    kills_args_3.push_back(MutableFunction::build("Curiosity"));
    kills_args_3.push_back(MutableFunction::build("Tuna"));

    const auto tuna_is_killed = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(MutableBinaryConnected::build(BinaryOperatorTypes::Disjunction,
                    MutablePredicate::build("Kills", std::move(kills_args_2)),
                    MutablePredicate::build("Kills", std::move(kills_args_3)))),
            symbol_repository);

    // Tuna is a cat.
    std::vector<std::unique_ptr<IMutableTerm>> cat_args_1;
    cat_args_1.push_back(MutableFunction::build("Tuna"));

    const auto tuna_is_cat = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(MutablePredicate::build("Cat", std::move(cat_args_1))),
            symbol_repository);

    // Cats are animals.
    std::vector<std::unique_ptr<IMutableTerm>> cat_args_2;
    cat_args_2.push_back(MutableVariable::build("x"));

    std::vector<std::unique_ptr<IMutableTerm>> animal_args_4;
    animal_args_4.push_back(MutableVariable::build("x"));

    const auto cats_are_animals = ExpressionFactory::build_sentence(
            MutableSentenceRoot::build(
                    MutableQuantified::build(QuantifierTypes::Universal, MutableVariable::build("x"),
                            MutableBinaryConnected::build(BinaryOperatorTypes::Implication,
                                    MutablePredicate::build("Cat", std::move(cat_args_2)),
                                    MutablePredicate::build("Animal", std::move(animal_args_4))))),
            symbol_repository);

    // Tell the KB the facts...
    prover->tell(*tuna_is_killed);
    prover->tell(*tuna_is_cat);
    prover->tell(*loves_all_animals);
    prover->tell(*kills_an_animal);
    prover->tell(*jack_loves_animals);
    prover->tell(*cats_are_animals);

    // Did Curiosity kill Tuna?
    std::vector<std::unique_ptr<IMutableTerm>> kills_args_4;
    kills_args_4.push_back(MutableFunction::build("Curiosity"));
    kills_args_4.push_back(MutableFunction::build("Tuna"));

    const auto result = prover->ask(
            MutableSentenceRoot::build(MutablePredicate::build("Kills", std::move(kills_args_4))));

    EXPECT_EQ(result.outcome, QueryResult::ConjectureStatus::Consistent);
}

} // namespace optifol
```

1.14.9 UnidirectionalUnificationTest.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Test unidirectional unification of FOL sentences and terms
 * @author Oliver Dixon
 * @date 2025-02-03
 * @version Development
 */
```

```cpp
#include <gtest/gtest.h>

#include "../IR/Sentences/Literal.hpp"
#include "../IR/SymbolRepository.hpp"
#include "../IR/Terms/Function.hpp"
#include "../IR/Terms/Variable.hpp"
#include "../Visitors/RegularTargets/Unification/UnificationVisitor.hpp"

namespace optifol
{

/**
 * @class UnidirectionalUnificationTest
 * @brief Tests the UnificationVisitor to verify that valid substitutions are constructed in the
 * Generalisation only to unify pairs of sentences.
 */
class UnidirectionalUnificationTest : public testing::Test
{
protected:
    std::unique_ptr<UnificationVisitor> unification_visitor;

    void SetUp() override
    {
        symbol_repository = std::make_shared<SymbolRepository>();
        unification_visitor = std::make_unique<UnificationVisitor>(symbol_repository);
    }

    template<typename TermType, class... CtorArgs>
    [[nodiscard]] const TermType *register_symbol(CtorArgs &&...ctor_args) const
    {
        return symbol_repository->add_symbol(
                std::make_unique<TermType>(std::forward<CtorArgs>(ctor_args)...));
    }

private:
    std::shared_ptr<SymbolRepository> symbol_repository;
};

/**
 * @brief Tests basic functionality of the UnidirectionalUnificationTest for a single pair of unifiable
 * literals with one applicable Function / Variable substitution.
 * @details
 *  <ul>
 *      <li>Generalisation: @f$ P \left( C\left(\right), x \right) @f$</li>
 *      <li>Instance: @f$ P \left( C\left(\right), D\left(\right) \right) @f$</li>
 *      <li>Expected substitutions: @f$ \left\{ x \mapsto D\left(\right) \right\} @f$</li>
 *  </ul>
 * @memberof UnidirectionalUnificationTest
 */
TEST_F(UnidirectionalUnificationTest, Positive_SingleBinding_FuncVar)
{
    const auto c = register_symbol<Function>("C");
    const auto d = register_symbol<Function>("D");
    const auto x = register_symbol<Variable>("x");

    const auto gen = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, x});
    const auto inst = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, d});

    EXPECT_TRUE(gen->accept(*unification_visitor, *inst));

    const Unifier expected_subs{{{x, d}}};
    EXPECT_EQ(*unification_visitor->observe_substitutions(), expected_subs);
}

/**
 * @brief Tests basic functionality of the UnidirectionalUnificationTest for a single pair of non-unifiable
 * literals, verifying that valid substitutions in the instance are refused.
 * @details
 *  <ul>
 *      <li>Generalisation: @f$ P \left( C\left(\right), x \right) @f$</li>
 *      <li>Instance: @f$ P \left( C\left(\right), D\left(\right) \right) @f$</li>
 *  </ul>
 * @memberof UnidirectionalUnificationTest
 */
TEST_F(UnidirectionalUnificationTest, Negative_SingleBinding_VarFunc)
{
    const auto c = register_symbol<Function>("C");
    const auto d = register_symbol<Function>("D");
    const auto x = register_symbol<Variable>("x");

    const auto gen = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, d});
```

```
    const auto inst = register_symbol<Literal>("P", std::vector<const IProcessedTerm *>{c, x});

    EXPECT_FALSE(gen->accept(*unification_visitor, *inst));
}

} // namespace optifol
```

1.15 UserTesting

1.15.1 Discovery

1.15.1.1 DiscoveryTestExecutable.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the DiscoveryTestExecutable storage object
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#include "DiscoveryTestExecutable.hpp"
#include "DiscoveryTestFixture.hpp"

namespace optifol
{

DiscoveryTestExecutable::DiscoveryTestExecutable(const Glib::ustring &executable_path) :
    Glib::ObjectBase("DiscoveryTestExecutable"),
    fixtures(Gio::ListStore<DiscoveryTestFixture>::create())
{
    property_name().set_value(executable_path);
}

DiscoveryTestExecutable::DiscoveryTestExecutable(const Glib::ustring &executable_path,
        BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder) :
    Glib::ObjectBase("DiscoveryTestExecutable"),
    StorageObjectBase(cobject, builder),
    fixtures(Gio::ListStore<DiscoveryTestFixture>::create())
{
    property_name().set_value(executable_path);
}

Glib::RefPtr<Gio::ListStore<DiscoveryTestFixture>> DiscoveryTestExecutable::get_fixture_model() const noexcept
{
    return fixtures;
}

Glib::RefPtr<DiscoveryTestFixture> DiscoveryTestExecutable::get_default_fixture() const
{
    return fixtures->get_item(0);
}

void DiscoveryTestExecutable::add_fixture(Glib::RefPtr<DiscoveryTestFixture> fixture) const
{
    fixtures->append(std::move(fixture));
}

bool DiscoveryTestExecutable::operator==(const DiscoveryTestExecutable &other) const
{
    return property_name().get_value() == other.property_name().get_value();
}

bool DiscoveryTestExecutable::operator==(const Glib::ustring &other_executable_path) const
{
    return property_name().get_value() == other_executable_path;
}

} // namespace optifol
```

1.15.1.2 DiscoveryTestExecutable.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the DiscoveryTestExecutable storage object
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#ifndef DISCOVERYTESTEXECUTABLE_HPP
#define DISCOVERYTESTEXECUTABLE_HPP

#include <giomm/liststore.h>

#include "../../Storage/StorageObjectBase.hpp"
#include "DiscoveryTestFixture.hpp"

namespace optifol
{

/**
 * @class DiscoveryTestExecutable
 * @brief Describes a testing framework-agnostic executable used to discover and model software unit fixtures
 * and tests thereof. Derived classes populate the executable with fixtures during or after
 * framework-dependent discovery, and consumers query the model from the base class.
 */
class DiscoveryTestExecutable : public StorageObjectBase
{
public:
    /**
     * @brief Retrieves the fixtures model to use in a Gtk::DropDown or Gtk::ListView.
     * @return A ref-counted pointer to the fixtures model, containing a single entry for each fixture within
     * the discovered executable.
     */
    Glib::RefPtr<Gio::ListStore<DiscoveryTestFixture>> get_fixture_model() const noexcept;

    /**
     * @brief Retrieves the first fixture object available within the discovered executable.
     * @return A ref-counted pointer to the first entry in the fixtures model.
     */
    Glib::RefPtr<DiscoveryTestFixture> get_default_fixture() const;

    /**
     * @brief Compare the test executable with an object of the same type.
     * @param other The other test executable.
     * @return Do the test executable objects refer to the same executable? Equality is determined by path.
     */
    bool operator==(const DiscoveryTestExecutable &other) const;

    /**
     * @brief Compare the test executable with the given exectable path.
     * @param other_executable_path The path of another executable on the file-system.
     * @return Does the test executable object and executable at the given path refer to the same executable?
     */
    bool operator==(const Glib::ustring &other_executable_path) const;

protected:
    /**
     * @brief Construct a new framework-agnostic test executable. Note that for the base constructor, no
     * discovery is performed and an empty fixture model is instantiated.
     * @param executable_path The path of the executable to be subject to discovery.
     */
    explicit DiscoveryTestExecutable(const Glib::ustring &executable_path);

    /**
     * @brief Construct a new framework-agnostic test executable. Note that for the base constructor, no
     * discovery is performed and an empty fixture model is instantiated.
     * @param executable_path The path of the executable to be subject to discovery.
     * @param cobject Glib C object.
     * @param builder Existing Gtk::Builder instance.
     */
    DiscoveryTestExecutable(const Glib::ustring &executable_path, BaseObjectType *cobject,
            const Glib::RefPtr<Gtk::Builder> &builder);

    /**
     * @brief Share a new fixture with the model. The shared fixture is appended to the internal model.
     * @param fixture The fixture to share.
```

```
     */
    void add_fixture(Glib::RefPtr<DiscoveryTestFixture> fixture) const;

private:
    const Glib::RefPtr<Gio::ListStore<DiscoveryTestFixture>> fixtures;
};

} // namespace optifol

template<>
struct std::hash<optifol::DiscoveryTestExecutable>
{
    using is_transparent = void;

    std::size_t operator()(const optifol::DiscoveryTestExecutable *object) const noexcept
    {
        return object->hash();
    }

    std::size_t operator()(const Glib::ustring &name) const noexcept
    {
        return std::hash<std::string>{}(name);
    }

    std::size_t operator()(const optifol::DiscoveryTestExecutable &object) const noexcept
    {
        return object.hash();
    }

    std::size_t operator()(
            const std::shared_ptr<optifol::DiscoveryTestExecutable> &shared_hashable) const noexcept
    {
        return shared_hashable->hash();
    }
};

#endif // DISCOVERYTESTEXECUTABLE_HPP
```

1.15.1.3 DiscoveryTestFixture.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

//
// Created by owd on 7/30/25.
//

#include "DiscoveryTestFixture.hpp"

namespace optifol
{

DiscoveryTestFixture::DiscoveryTestFixture(const Glib::ustring &name) :
    Glib::ObjectBase("DiscoveryTestFixture")
{
    property_name().set_value(name);
}

DiscoveryTestFixture::DiscoveryTestFixture(
        const Glib::ustring &name, BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder) :
    Glib::ObjectBase("DiscoveryTestFixture"),
    StorageObjectBase(cobject, builder)
{
    property_name().set_value(name);
}

void DiscoveryTestFixture::add_test(const Glib::ustring &name) const
{
    test_names->append(name);
}

Glib::RefPtr<Gtk::StringList> DiscoveryTestFixture::get_test_model() noexcept
{
    return test_names;
}

} // namespace optifol
```

1.15.1.4 DiscoveryTestFixture.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

//
// Created by owd on 7/30/25.
//

#ifndef DISCOVERYTESTFIXTURE_HPP
#define DISCOVERYTESTFIXTURE_HPP

#include <gtkmm/stringlist.h>
#include "../../Storage/StorageObjectBase.hpp"

namespace optifol
{

class DiscoveryTestFixture : public StorageObjectBase
{
public:
    explicit DiscoveryTestFixture(const Glib::ustring &name);

    DiscoveryTestFixture(
            const Glib::ustring &name, BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);

    void add_test(const Glib::ustring &name) const;

    Glib::RefPtr<Gtk::StringList> get_test_model() noexcept;

private:
    Glib::RefPtr<Gtk::StringList> test_names = Gtk::StringList::create();
};

} // namespace optifol

#endif // DISCOVERYTESTFIXTURE_HPP
```

1.15.1.5 GoogleTestDiscoveryExecutable.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the GoogleTestDiscoveryExecutable discovery executable backend
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#include <cassert>
#include <fstream>
#include <rapidjson/istreamwrapper.h>

#include "../../Exceptions/ParseError.hpp"
#include "../Logging.hpp"
#include "DiscoveryTestFixture.hpp"
#include "GoogleTestDiscoveryExecutable.hpp"

namespace optifol
{

const log4cxx::LoggerPtr GoogleTestDiscoveryExecutable::logger =
        Logging::get_logger({"UserTesting", "Discovery", "GoogleTest"});

GoogleTestDiscoveryExecutable::GoogleTestDiscoveryExecutable(const Glib::ustring &executable_path) :
    Glib::ObjectBase("GoogleTestDiscoveryExecutable"),
    DiscoveryTestExecutable(executable_path)
{
    start_discovery();
}

GoogleTestDiscoveryExecutable::GoogleTestDiscoveryExecutable(const Glib::ustring &executable_path,
        BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder) :
```

```
    Glib::ObjectBase("GoogleTestDiscoveryExecutable"),
    DiscoveryTestExecutable(executable_path, cobject, builder)
{
    start_discovery();
}

GoogleTestDiscoveryExecutable::~GoogleTestDiscoveryExecutable()
{
    if (discovery_executor != nullptr)
        logger->error("Google Test discovery executable for \"" + property_name().get_value() +
                "\" is being "
                "destructed with an unhandled subprocess. Discovery will be incomplete.");

    if (discovery_tmp_file_path.has_value())
        logger->error("Google Test discovery executable for \"" + property_name().get_value() +
                "\" is being "
                "destructed with an unresolved temporary file at \"" +
                *discovery_tmp_file_path + "\". Discovery will be incomplete.");
}

void GoogleTestDiscoveryExecutable::start_discovery()
{
    const auto current_time_ms = std::chrono::duration_cast<std::chrono::milliseconds>(
            std::chrono::system_clock::now().time_since_epoch());

    discovery_tmp_file_path.emplace(
            "/tmp/google_test_discovery." + std::to_string(current_time_ms.count()) + ".json");
    discovery_executor = std::make_unique<ProcessExecutor>("", // Current working directory.
            std::vector<std::string>{property_name().get_value(), "--gtest_list_tests",
                    "--gtest_output=json:" + *discovery_tmp_file_path},
            std::vector<std::string>{},
            sigc::mem_fun(*this, &GoogleTestDiscoveryExecutable::discovery_done_callback));
}

void GoogleTestDiscoveryExecutable::discovery_done_callback(const int exit_code) noexcept
{
    assert(discovery_tmp_file_path.has_value());

    // In any case, remove the completed sub-process.
    discovery_executor.reset();

    if (exit_code != 0) {
        logger->error("Discovery sub-process exited with non-zero exit code; no parsing attempted.");
        std::remove(discovery_tmp_file_path->c_str());
        discovery_tmp_file_path.reset();
        return;
    }

    try {
        // Open the temporary file for recovery of test information.
        std::ifstream discovery_file_stream;
        discovery_file_stream.exceptions(discovery_file_stream.exceptions() | std::ios::badbit);
        discovery_file_stream.open(*discovery_tmp_file_path);

        // Parse the JSON payload generated by the Google Test sub-process.
        rapidjson::IStreamWrapper json_stream{discovery_file_stream};
        rapidjson::Document document;
        document.ParseStream(json_stream); // May invoke RAPIDJSON_PARSE_ERROR_NORETURN.
        parse_json_payload(document);
    } catch (const ParseError &parse_error) {
        logger->error(
                "Parse failed for Google Test discovery executable \"" + property_name().get_value() + "\".");
        logger->error(parse_error.what());
    } catch (const std::ios_base::failure &stream_error) {
        logger->error("Could not open file \"" + *discovery_tmp_file_path + "\" due to system error.");
        logger->error(stream_error.what());
    }

    // Remove the temporary file and reset the variable.
    if (std::remove(discovery_tmp_file_path->c_str()) != 0)
        logger->warn("Could not remove temporary discovery file \"" + *discovery_tmp_file_path + "\".");

    discovery_tmp_file_path.reset();

    assert(discovery_executor == nullptr);
    assert(discovery_tmp_file_path.has_value() == false);
}
// ReSharper disable once CppDFAUnreachableFunctionCall - False positive. Called from discovery_done_callback.
void GoogleTestDiscoveryExecutable::parse_json_payload(const rapidjson::Document &document) const
{
    const auto &fixtures = document.FindMember("testsuites");
```

```
    if (fixtures == document.MemberEnd() || fixtures->value.IsArray() == false)
        throw ParseError("Test suites array not found.");

    for (const auto &fixture: fixtures->value.GetArray()) {
        if (fixture.IsObject() == false)
            throw ParseError("Fixture located, but is not an object.");

        const auto fixture_name = fixture.FindMember("name");
        if (fixture_name == fixture.MemberEnd() || fixture_name->value.IsString() == false)
            throw ParseError("Fixture name not found.");

        const auto fixture_tests = fixture.FindMember("testsuite");
        if (fixture_tests == fixture.MemberEnd() || fixture_tests->value.IsArray() == false)
            throw ParseError("Fixture tests not present for \"" +
                    std::string(fixture_name->value.GetString()) + "\".");

        const auto modelled_fixture =
                Glib::make_refptr_for_instance(new DiscoveryTestFixture(fixture_name->value.GetString()));

        for (const auto &test: fixture_tests->value.GetArray()) {
            if (fixture.IsObject() == false)
                throw ParseError("Test located in \"" + std::string(fixture_name->value.GetString()) +
                        "\", but is not an object.");

            const auto test_name = test.FindMember("name");
            if (test_name == test.MemberEnd() || test_name->value.IsString() == false)
                throw ParseError("Test located in \"" + std::string(fixture_name->value.GetString()) +
                        "\", but it does not have a declared name.");

            modelled_fixture->add_test(test_name->value.GetString());
        }

        add_fixture(modelled_fixture);
    }
}

} // namespace optifol
```

1.15.1.6 GoogleTestDiscoveryExecutable.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the GoogleTestDiscoveryExecutable discovery executable backend
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#ifndef GOOGLETESTDISCOVERYEXECUTABLE_HPP
#define GOOGLETESTDISCOVERYEXECUTABLE_HPP

#include <rapidjson/document.h>

#include "../../GUI/ProcessExecutor.hpp"
#include "DiscoveryTestExecutable.hpp"

namespace optifol
{

/**
 * @class GoogleTestDiscoveryExecutable
 * @brief Provides a non-abstract implementation of DiscoveryTestExecutable for the Google Test unit-testing
 * framework.
 *
 * @details Google Test executables are queried for discovery via the asynchronous non-streaming
 * ProcessExecutor. The executable generates a JSON-formatted structure of fixtures and test names which is
 * parsed and stored in the internal structure. The JSON is temporarily stored on disk and is deleted after
 * parsing. As an RAII class, discovery (and subsequent subprocess invocation) is performed during
 * construction.
 */
class GoogleTestDiscoveryExecutable : public DiscoveryTestExecutable
{
public:
    /**
```

```
     * @brief Construct a GoogleTestDiscoveryExecutable for a Google Test executable at the given path.
     * @param executable_path The file-system path, relative to the CWD of the executable, to the Google Test
     *  executable.
     */
    explicit GoogleTestDiscoveryExecutable(const Glib::ustring &executable_path);

    /**
     * @brief Construct a GoogleTestDiscoveryExecutable for a Google Test executable at the given path.
     * @param executable_path The file-system path, relative to the CWD of the executable, to the Google Test
     *  executable.
     * @param cobject Glib C object
     * @param builder Gtk::Builder object
     */
    GoogleTestDiscoveryExecutable(const Glib::ustring &executable_path, BaseObjectType *cobject,
            const Glib::RefPtr<Gtk::Builder> &builder);

    /**
     * @brief Destruct the GoogleTestDiscoveryExecutable, logging errors if there is an unresolved
     * sub-process.
     */
    ~GoogleTestDiscoveryExecutable() override;

private:
    /**
     * @brief Initiates discovery process on the loaded Google Test executable. A temporary file is created
     * for the asynchronously launched process to write a JSON-formatted structure describing the test
     * structure.
     */
    void start_discovery();

    /**
     * @brief Handles the end of the Google Test discovery sub-process: parses the JSON into the fixture/test
     * model of the DiscoveryTestExecutable base class, and and deletes the temporary file.
     * @param exit_code The exit code of the executable.
     * @pre @ref discovery_tmp_file_path is populated.
     * @post @ref discovery_tmp_file_path is unpopulated, incdicating removal of the temporary file.
     * @post @ref discovery_executor is unpopulated, indicating completion of the G-Test sub-process.
     */
    void discovery_done_callback(int exit_code) noexcept;

    /**
     * @brief Parse the JSON payload produced by Google Test, parsed by RapidJSON into the given document.
     * @param document The parsed DOM of the JSON payload.
     * @throws ParseError if the document schema was invalid.
     */
    void parse_json_payload(const rapidjson::Document &document) const;

    static const log4cxx::LoggerPtr logger;

    std::unique_ptr<ProcessExecutor> discovery_executor;
    std::optional<std::string> discovery_tmp_file_path;
};

} // namespace optifol

#endif // GOOGLETESTDISCOVERYEXECUTABLE_HPP
```

1.15.1.7 TestSpecificationEntry.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the TestSpecificationEntry storage object
 * @author Oliver Dixon
 * @date 2025-07-31
 * @version Development
 */

#include "TestSpecificationEntry.hpp"

namespace optifol
{

TestSpecificationEntry::TestSpecificationEntry() :
    Glib::ObjectBase("TestSpecificationEntry"),
```

```
    executable(*this, "TestSpecificationEntry-executable"),
    fixture(*this, "TestSpecificationEntry-fixture")
{
    setup_sync_callbacks();
}

TestSpecificationEntry::TestSpecificationEntry(
        BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder) :
    Glib::ObjectBase("TestSpecificationEntry"),
    StorageObjectBase(cobject, builder),
    executable(*this, "TestSpecificationEntry-executable"),
    fixture(*this, "TestSpecificationEntry-fixture")
{
    setup_sync_callbacks();
}

TestSpecificationEntry::TestSpecificationEntry(const TestSpecificationEntry &template_entry) :
    Glib::ObjectBase("TestSpecificationEntry"),
    executable(*this, "TestSpecificationEntry-executable", template_entry.property_executable().get_value()),
    fixture(*this, "TestSpecificationEntry-fixture", template_entry.property_fixture().get_value())
{
    property_name().set_value(template_entry.property_name().get_value());
}

Glib::PropertyProxy<Glib::RefPtr<DiscoveryTestExecutable>> TestSpecificationEntry::property_executable()
{
    return executable.get_proxy();
}

Glib::PropertyProxy<Glib::RefPtr<DiscoveryTestFixture>> TestSpecificationEntry::property_fixture()
{
    return fixture.get_proxy();
}

Glib::PropertyProxy_ReadOnly<Glib::RefPtr<DiscoveryTestExecutable>>
TestSpecificationEntry::property_executable() const
{
    return executable.get_proxy();
}

Glib::PropertyProxy_ReadOnly<Glib::RefPtr<DiscoveryTestFixture>>
TestSpecificationEntry::property_fixture() const
{
    return fixture.get_proxy();
}

void TestSpecificationEntry::setup_sync_callbacks()
{
    property_executable().signal_changed().connect(
            [this] { property_fixture().set_value(executable.get_value()->get_default_fixture()); });
}

} // namespace optifol
```

1.15.1.8 TestSpecificationEntry.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the TestSpecificationEntry storage object
 * @author Oliver Dixon
 * @date 2025-07-31
 * @version Development
 */

#ifndef TESTSPECIFICATIONENTRY_HPP
#define TESTSPECIFICATIONENTRY_HPP

#include "../../Storage/StorageObjectBase.hpp"
#include "DiscoveryTestExecutable.hpp"

namespace optifol
{

/**
```

```
 * @class TestSpecificationEntry
 * @brief Provides a GObject container for description of a test specification. A test specification can be
 * used to construct a Test and holds a test executable, a fixture, and a test name.
 * @details Test specifications are useful during the <i>discovery</i> phase of user testing. In particular,
 * the user selects a test executable provided by a unit-testing framework (e.g. Google Test), the executable
 * is queried either from the file-system or from an internal cache, and string models are constructed to
 * detain the available fixtures and tests thereof. These properties are encoded by classes
 * DiscoveryTestExecutable and DiscoveryTestFixture respectively, and held under shared ownership by the test
 * specification.
 */
class TestSpecificationEntry : public StorageObjectBase
{
public:
    /**
     * @brief Construct a new TestSpecificationEntry to register as a GType with empty properties.
     */
    TestSpecificationEntry();

    /**
     * @brief Construct a new TestSpecificationEntry to register as a GType with empty properties and an
     * existing C object.
     * @param cobject Existing C object
     * @param builder Existing source Gtk::Builder
     */
    TestSpecificationEntry(BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);

    /**
     * @brief Copy-construct a TestSpecificationEntry from the given template entry.
     * @param template_entry The entry to clone.
     */
    TestSpecificationEntry(const TestSpecificationEntry &template_entry);

    /**
     * @brief Retrieves a read-write proxy for the DiscoveryTestExecutable executable property.
     * @return A read-write proxy for the executable.
     */
    [[nodiscard]] Glib::PropertyProxy<Glib::RefPtr<DiscoveryTestExecutable>> property_executable();

    /**
     * @brief Retrieves a read-write proxy for the DiscoveryTestFixture fixture property.
     * @return A read-write proxy for the fixture.
     */
    [[nodiscard]] Glib::PropertyProxy<Glib::RefPtr<DiscoveryTestFixture>> property_fixture();

    /**
     * @brief Retrieves a read-only proxy for the DiscoveryTestExecutable executable property.
     * @return A read-only proxy for the executable.
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::RefPtr<DiscoveryTestExecutable>>
    property_executable() const;

    /**
     * @brief Retrieves a read-only proxy for the DiscoveryTestFixture fixture property.
     * @return A read-only proxy for the fixture.
     */
    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::RefPtr<DiscoveryTestFixture>> property_fixture() const;

private:
    /**
     * @brief Establish callbacks such that the internal intra-model state stays correctly synchronised.
     * @details In particular, changes in the @ref executable will unidirectionally propagate to the @ref
     * fixture.
     */
    void setup_sync_callbacks();

    /**
     * @brief The discovered executable shared-ownership reference property.
     * @details This property shares ownership for a DiscoveryTestExecutable object. Shared ownership is
     * necessary here; multiple TestSpecificationEntry objects are likely to refer to the same
     * DiscoveryTestExecutable. Due to this, it is likely that callers (c.f. ManageTestsPopover) will want to
     * maintain some sort of set-based cache and provide references. Lifetime guarantees cannot be provided on
     * the cache, so shared ownership is necessary.
     */
    Glib::Property<Glib::RefPtr<DiscoveryTestExecutable>> executable;

    /**
     * @brief The discovered fixture shared-ownership reference property.
     * @details This property shares ownership for a DiscoveryTestFixture object. Shared ownership is
     * unfortunately necessary here. The DiscoveryTestFixture will, under defined operating conditions, be a
     * member of the fixtures detained by @ref executable. Although lifetime guarantees can be placed on the
     * DiscoveryTestExecutable, Glib does not guarantee stable pointers within the internal Gio::ListModel;
```

```
     * hence we need to claim ownership locally.
     */
    Glib::Property<Glib::RefPtr<DiscoveryTestFixture>> fixture;
};

} // namespace optifol

#endif // TESTSPECIFICATIONENTRY_HPP
```

1.15.2 Execution

1.15.2.1 PartialTestResult.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

//
// Created by owd on 8/10/25.
//

#include "PartialTestResult.hpp"

namespace optifol
{

PartialTestResult::PartialTestResult(std::string file, const std::size_t line, const std::string &message) :
    Glib::ObjectBase("PartialTestResult"),
    file(*this, "PartialTestResult-file", std::move(file)),
    line(*this, "PartialTestResult-line", static_cast<guint>(line))
{
    property_name().set_value(message);
}

PartialTestResult::PartialTestResult(std::string file, const std::size_t line, const std::string &message,
        BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder) :
    Glib::ObjectBase("PartialTestResult"),
    StorageObjectBase(cobject, builder),
    file(*this, "PartialTestResult-file", std::move(file)),
    line(*this, "PartialTestResult-line", static_cast<guint>(line))
{
    property_name().set_value(message);
}

Glib::RefPtr<Gtk::TreeListModel> PartialTestResult::get_tests_tree() const noexcept
{
    return nullptr;
}

Glib::RefPtr<Gtk::TreeListModel> PartialTestResult::get_results_tree() const noexcept
{
    return nullptr;
}

Glib::PropertyProxy<Glib::ustring> PartialTestResult::property_file()
{
    return file.get_proxy();
}

Glib::PropertyProxy<guint> PartialTestResult::property_line()
{
    return line.get_proxy();
}

Glib::PropertyProxy_ReadOnly<Glib::ustring> PartialTestResult::property_file() const
{
    return file.get_proxy();
}

Glib::PropertyProxy_ReadOnly<guint> PartialTestResult::property_line() const
{
    return line.get_proxy();
}

} // namespace optifol
```

1.15.2.2 PartialTestResult.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

//
// Created by owd on 8/10/25.
//

#ifndef OPTIFOL_PARTIALTESTRESULT_HPP
#define OPTIFOL_PARTIALTESTRESULT_HPP

#include "../../Storage/StorageObjectBase.hpp"
#include "../Modelling/ITestModelNode.hpp"

namespace optifol
{

class PartialTestResult : public StorageObjectBase,
                          public ITestModelNode
{
public:
    PartialTestResult(std::string file, std::size_t line, const std::string &message);

    PartialTestResult(std::string file, std::size_t line, const std::string &message, BaseObjectType *cobject,
            const Glib::RefPtr<Gtk::Builder> &builder);

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_tests_tree() const noexcept override;

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_results_tree() const noexcept override;

    [[nodiscard]] Glib::PropertyProxy<Glib::ustring> property_file();

    [[nodiscard]] Glib::PropertyProxy<guint> property_line();

    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::ustring> property_file() const;

    [[nodiscard]] Glib::PropertyProxy_ReadOnly<guint> property_line() const;

private:
    Glib::Property<Glib::ustring> file;
    Glib::Property<guint> line;
};

} // namespace optifol

#endif // OPTIFOL_PARTIALTESTRESULT_HPP
```

1.15.2.3 TestExecutable.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the abstract TargetTestExecutableBase
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#include "TestExecutable.hpp"

namespace optifol
{

TestExecutable::TestExecutable(const Glib::ustring &executable_path) :
    Glib::ObjectBase("TestExecutable")
{
    property_name().set_value(executable_path);
}

TestExecutable::TestExecutable(const Glib::ustring &executable_path, BaseObjectType *cobject,
        const Glib::RefPtr<Gtk::Builder> &builder) :
    Glib::ObjectBase("TestExecutable"),
```

```
    StorageObjectBase(cobject, builder)
{
    property_name().set_value(executable_path);
}

bool TestExecutable::operator==(const TestExecutable &other) const noexcept
{
    return property_name().get_value() == other.property_name().get_value();
}

} // namespace optifol
```

1.15.2.4 TestExecutable.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the abstract TargetTestExecutableBase
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#ifndef TARGETTESTEXECUTABLE_HPP
#define TARGETTESTEXECUTABLE_HPP

#include <giomm/filemonitor.h>
#include <unordered_set>

#include "../../Storage/StorageObjectBase.hpp"

namespace optifol
{

class Requirement;
class DiscoveryTestExecutable;
class TestResult;

/**
 * @class TestExecutable
 * @brief Describes a framework-agnostic test executable provided by a software unit-testing system. The
 * TestExecutable is a StorageObjectBase such that its main attributes are accessed via the Glib types and
 * properties system.
 * @see DiscoveryTestExecutable for the non-runnable discovery equivalent.
 */
class TestExecutable : public StorageObjectBase
{
public:
    /**
     * @brief Create a new TargetTestExecutableBase with a name.
     * @param executable_path The path of the test executable.
     */
    explicit TestExecutable(const Glib::ustring &executable_path);

    /**
     * @brief TargetTestExecutableBase
     * @param executable_path The path of the test executable.
     * @param cobject Glib C object
     * @param builder Gtk::Builder template
     */
    TestExecutable(const Glib::ustring &executable_path, BaseObjectType *cobject,
            const Glib::RefPtr<Gtk::Builder> &builder);

    bool operator==(const TestExecutable &other) const noexcept;

private:
    Glib::RefPtr<Gio::FileMonitor> file_monitor;
    std::unordered_set<std::shared_ptr<TestResult>> received_blob;
    bool is_outdated = false;
};

} // namespace optifol

#endif // TARGETTESTEXECUTABLE_HPP
```

1.15.2.5 TestResult.cpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the TestResult results storage
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#include "TestResult.hpp"

#include <cassert>
#include <utility>

namespace optifol
{

TestResult::TestResult(Glib::ustring test_name, const bool passed, const std::size_t execution_time,
        std::vector<Glib::RefPtr<PartialTestResult>> &&partial_results) :
    test_name(std::move(test_name)),
    passed(passed),
    execution_time(execution_time)
{
    for (const auto &partial_result: partial_results)
        this->partial_results->append(partial_result);
}

std::size_t TestResult::hash() const noexcept
{
    return hash_combine(std::hash<std::string>{}(fixture_name), std::hash<std::string>{}(test_name));
}

Glib::RefPtr<Gtk::TreeListModel> TestResult::get_tests_tree() const noexcept
{
    return nullptr;
}

Glib::RefPtr<Gtk::TreeListModel> TestResult::get_results_tree() const noexcept
{
    return partial_results_tree;
}

void TestResult::populate_test_fixture_name(const std::string &incoming_fixture_name)
{
    fixture_name = incoming_fixture_name;
}

Glib::ustring TestResult::copy_test_name() const noexcept
{
    return test_name;
}

bool TestResult::has_passed() const noexcept
{
    return passed;
}

std::size_t TestResult::get_execution_time() const noexcept
{
    return execution_time;
}

Glib::ustring TestResult::copy_fixture_name() const noexcept
{
    return fixture_name;
}

bool TestResult::operator==(const TestResult &other) const noexcept
{
    return test_name == other.test_name && fixture_name == other.fixture_name;
}

bool TestResult::operator==(const std::pair<Glib::ustring, Glib::ustring> &names) const noexcept
{
```

```
    return names.first == fixture_name && names.second == test_name;
}

} // namespace optifol
```

1.15.2.6 TestResult.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the TestResult results storage
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#ifndef TESTRESULT_HPP
#define TESTRESULT_HPP

#include <giomm/liststore.h>
#include <glibmm/ustring.h>
#include <optional>
#include <string>
#include <vector>

#include "../../IHashable.hpp"
#include "../Modelling/ITestModelNode.hpp"
#include "PartialTestResult.hpp"

namespace optifol
{

/**
 * @class TestResult
 * @brief Describes a single result provided by a software unit-testing framework. The TestResult is agnostic
 * to the framework, but does encode the test name, test fixture name, and executable path for which the
 * result is relevant.
 */
class TestResult : public IHashable,
                   public ITestModelNode
{
public:
    /**
     * @brief Create a new TestResult with the given name, result, execution time, and partial results.
     * @details Note that the test fixture name is not specified upon construction. This is due to a
     * requirement on many backend parsers that the fixture object is parsed after its constituent test
     * results. Therefore the fixture name is specified after construction with @ref
     * populate_test_fixture_name.
     * @param test_name The name of the individual test.
     * @param passed Did the test pass?
     * @param execution_time Running execution time of the test.
     * @param partial_results Partial results generated by the testing framework.
     */
    TestResult(Glib::ustring test_name, bool passed, std::size_t execution_time,
            std::vector<Glib::RefPtr<PartialTestResult>> &&partial_results = {});

    /**
     * @copybrief IHashable::hash
     * @return The hashcode of the TestResult object
     * @warning This hash function is incomplete. It will only uniquely identify the Test with which the
     * result is associated, according to the test name and fixture name. It does not consider the fabric of
     * the result.
     */
    [[nodiscard]] std::size_t hash() const noexcept override;

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_tests_tree() const noexcept override;

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_results_tree() const noexcept override;

    /**
     * @brief Endow the TestResult with the name of the fixture of the Test to which the TestResult relates.
     * @param incoming_fixture_name The fixture name.
     */
    void populate_test_fixture_name(const std::string &incoming_fixture_name);
```

```
    /**
     * @brief Get a copy of the Test name associated with the TestResult.
     * @return The name of the Test to which the TestResult relates.
     */
    [[nodiscard]] Glib::ustring copy_test_name() const noexcept;

    /**
     * @brief Queries the binary result.
     * @return Did the test pass?
     */
    [[nodiscard]] bool has_passed() const noexcept;

    /**
     * @brief Queries the test execution time.
     * @return Duration time for execution of the test.
     */
    [[nodiscard]] std::size_t get_execution_time() const noexcept;

    /**
     * @brief Get a copy of the fixture name of the Test associated with the TestResult.
     * @return The name of the fixture to which the TestResult relates.
     */
    [[nodiscard]] Glib::ustring copy_fixture_name() const noexcept;

    /**
     * @brief Compare two TestResult objects using only the associated test name and fixture name.
     * @param other The other TestResult.
     * @return Are the two TestResult objects relating to the same Test?
     */
    bool operator==(const TestResult &other) const noexcept;

    /**
     * @brief Compare a TestResult object with an explicit test name and fixture name.
     * @param names A @ref std::pair encoding of the fixture name followed by the test name.
     * @return Is the TestResult for the fixture-test Test specified by the pair?
     */
    bool operator==(const std::pair<Glib::ustring, Glib::ustring> &names) const noexcept;

private:
    const Glib::ustring test_name;
    Glib::ustring fixture_name;

    const bool passed;
    const std::size_t execution_time;

    const Glib::RefPtr<Gio::ListStore<PartialTestResult>> partial_results =
            Gio::ListStore<PartialTestResult>::create();
    const Glib::RefPtr<Gtk::TreeListModel> partial_results_tree =
            Gtk::TreeListModel::create(partial_results, &ITestModelNode::get_given_results_tree, true);
};

} // namespace optifol

template<>
struct std::hash<optifol::TestResult>
{
    using is_transparent = void;

    std::size_t operator()(const optifol::TestResult *object) const noexcept
    {
        return object->hash();
    }

    std::size_t operator()(const std::pair<Glib::ustring, Glib::ustring> &names) const noexcept
    {
        return optifol::IHashable::hash_combine(
                std::hash<std::string>{}(names.first), std::hash<std::string>{}(names.second));
    }

    std::size_t operator()(const optifol::TestResult &object) const noexcept
    {
        return object.hash();
    }

    std::size_t operator()(const std::shared_ptr<optifol::TestResult> &shared_hashable) const noexcept
    {
        return shared_hashable->hash();
    }
};

#endif // TESTRESULT_HPP
```

1.15.2.7 PayloadManagement

1.15.2.7.1 GoogleTestLexer.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Google Test payload lexer base class
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#ifndef GOOGLETESTLEXER_HPP
#define GOOGLETESTLEXER_HPP

#pragma clang diagnostic push
#pragma ide diagnostic ignored "OCUnusedStructInspection"
#pragma ide diagnostic ignored "NotImplementedFunctions"
#pragma ide diagnostic ignored "OCUnusedGlobalDeclarationInspection"

#include "GoogleTestParser.hpp"

#ifndef __FLEX_LEXER_H
#undef yyFlexLexer
#define yyFlexLexer GoogleTestFlexLexer
#include <FlexLexer.h>
#endif

namespace optifol
{

/**
 * @class GoogleTestLexer
 * @brief Base class for the Flex-generated C++ lexer to lex Google Test results payloads.
 * @see GoogleTestParser for the parsing dual.
 */
class GoogleTestLexer : public yyFlexLexer
{
public:
    /**
     * @brief Construct a new lexer object for lexing Google Test result payloads.
     * @param yy_in The input stream.
     * @param yy_out The output stream.
     */
    GoogleTestLexer(std::istream &yy_in, std::ostream &yy_out) :
        yyFlexLexer(yy_in, yy_out)
    {
    }

    /**
     * @brief Lex the next token on the input stream and
     * @param yylval The lexed token type output parameter
     * @return Flex error state
     */
    int lex(GoogleTestParser::semantic_type *yylval);
};

} // namespace optifol

#pragma clang diagnostic pop

#endif
```

1.15.2.7.2 GoogleTestLexer.l

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

%{
#include "GoogleTestParser.hpp"
#include "GoogleTestLexer.hpp"
```

```
#undef YY_DECL
#define YY_DECL int optifol::GoogleTestLexer::lex(optifol::GoogleTestParser::semantic_type *yylval)
%}

%option c++ interactive noyywrap noyylineno
%option yyclass="GoogleTestLexer"
%option prefix="GoogleTest"

whitespace ([ \t\n]+)

%%

[^\t\n=&]+ {
    yylval->emplace<std::string>(YYText());
    return GoogleTestParser::token::Literal;
}

"gtest_streaming_protocol_version=" return GoogleTestParser::token::ProtocolVersion;
"&elapsed_time=" return GoogleTestParser::token::ElapsedTime;
"&passed=" return GoogleTestParser::token::Passed;
"&name=" return GoogleTestParser::token::Name;

"event=TestProgramStart" return GoogleTestParser::token::ProgramStart;
"event=TestProgramEnd" return GoogleTestParser::token::ProgramEnd;

"event=TestIterationStart" return GoogleTestParser::token::IterationStart;
"&iteration=" return GoogleTestParser::token::IterationCount;
"event=TestIterationEnd" return GoogleTestParser::token::IterationEnd;

"event=TestCaseStart" return GoogleTestParser::token::TestCaseStart;
"event=TestCaseEnd" return GoogleTestParser::token::TestCaseEnd;

"event=TestStart" return GoogleTestParser::token::TestStart;
"event=TestPartResult" return GoogleTestParser::token::TestPartial;
"&file=" return GoogleTestParser::token::File;
"&line=" return GoogleTestParser::token::Line;
"&message=" return GoogleTestParser::token::Message;
"event=TestEnd" return GoogleTestParser::token::TestEnd;

<<EOF>> return GoogleTestParser::token::End;
{whitespace} /* Ignore whitespace */

%%
```

1.15.2.7.3 GoogleTestListener.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Google Test TCP socket listener
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#include <giomm/inetsocketaddress.h>
#include <giomm/resource.h>
#include <giomm/socketlistener.h>

#include "../../../Logging.hpp"
#include "GoogleTestListener.hpp"

namespace optifol
{

const log4cxx::LoggerPtr GoogleTestListener::logger =
        Logging::get_logger({"UserTesting", "Network", "GoogleTest"});

GoogleTestListener::GoogleTestListener(const guint16 port_number) :
    parser(&lexer, sigc::mem_fun(*this, &TestListenerBase::accept_result))
{
    try {
        const auto address = Gio::InetAddress::create_loopback(Gio::SocketFamily::IPV4);
        const auto socket_address = Gio::InetSocketAddress::create(address, port_number);
```

```
        Glib::RefPtr<Gio::SocketAddress> effective_address;

        if (listener->add_address(socket_address, Gio::Socket::Type::STREAM, Gio::Socket::Protocol::TCP,
                    effective_address) == false)
            throw Gio::ResourceError(
                    Gio::ResourceError::INTERNAL, "Could not bind to " + socket_address->to_string());

        listener->accept_async(sigc::mem_fun(*this, &GoogleTestListener::connection_callback));
        logger->info("Opened listening TCP socket on " + effective_address->to_string());
    } catch (const Glib::Error &exception) {
        logger->error("Cannot open TCP socket for listening.");
        logger->error(exception.what());
        throw;
    }
}

void GoogleTestListener::connection_callback(const Glib::RefPtr<Gio::AsyncResult> &result) noexcept
{
    logger->debug("Connection callback triggered; attempting to decode client and read bytes.");

    try {
        const auto connection = listener->accept_finish(result);
        logger->info("Accepted TCP connection from " + connection->get_remote_address()->to_string());

        const auto input_stream = connection->get_input_stream();

        /*
         * We can make some assertions on the full results string, so std::string::append is likely more
         * performant than std::ostringstream::operator<<. See https://stackoverflow.com/questions/19844858.
         * In particular, we know the following:
         *
         *  1. The parser is non-streaming, so we need to have the entire input stored before passing it to
         * the lexer and parser routines. When the TCP packets arrive in chunks of
         * GoogleTestListener::temp_buffer_size bytes, they must be collated into a single source; and
         *
         *  2. The collated buffer will grow by factors of GoogleTestListener::temp_buffer_size per append
         * operation, so we can do allocation ahead of time.
         */

        std::string results_string;
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunsafe-buffer-usage"
        /*
         * Our buffer access is guaranteed to be safe. Access is performed at two points:
         *
         *  1. Gio::InputStream::read mutates the buffer in the maximal half-open interval [0,
         * sizeof(buffer)). This is guaranteed by documented constraints on the read function; and
         *
         *  2. We manually append a NULL-terminator at the position indicated by the return value of the read
         * function. This is further guaranteed to be within the half-open interval [0, sizeof(buffer)), hence
         * the access can occur at most sizeof(buffer) - 1 bytes into the array.
         *
         * The use of an "unsafe" C construct is enforced by the GioMM library.
         */
        char buffer[temp_buffer_size];

        gssize bytes_read = 0;
        while ((bytes_read = input_stream->read(buffer, sizeof(buffer) - 1)) > 0) {
            /*
             * We rather do a single worst-case allocation of the entire temporary buffer than lots of small
             * allocations in the actual number of bytes read. In Google Test streaming, packets are often
             * small and numerous.
             */
            results_string.reserve(results_string.capacity() + temp_buffer_size);
            buffer[bytes_read] = '\0';
            logger->trace("Read " + std::to_string(bytes_read) + " from input stream of last connection.");
            results_string.append(buffer);
        }
#pragma clang diagnostic pop

        listener->accept_async(sigc::mem_fun(*this, &GoogleTestListener::connection_callback));

        /*
         * Push the collated input into a lexer stream, lex and parse, and execute the closed callback to
         * indicate that the client stopped sending data and the connection was closed.
         */
        lexer_input_stream.str(results_string);
        parser.parse();
    } catch (const Glib::Error &exception) {
        logger->error("Cannot accept or read from client on TCP socket.");
        logger->error(exception.what());
```

```
    } catch (const SemanticException &) {
        logger->error("Semantic error during parse: the packets were received and complied with the protocol "
                      "schema, "
                      "but were meaningless.");
    } catch (const ParseError &) {
        logger->error("Parse error during reception: the packets were received, but did not comply with the "
                      "schema.");
    }
}

} // namespace optifol
```

1.15.2.7.4 GoogleTestListener.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Google Test TCP socket listener
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#ifndef GOOGLETESTLISTENER_HPP
#define GOOGLETESTLISTENER_HPP

#include <giomm/socketlistener.h>
#include <log4cxx/logger.h>

#include "GoogleTestLexer.hpp"
#include "TestListenerBase.hpp"

namespace optifol
{

class TestResult;

/**
 * @class GoogleTestListener
 * @brief Accept incoming packets from the Google Test testing framework, parse the payloads into TestResult
 * objects, and invoke the listening callbacks.
 *
 * @details The constructed class opens a non-blocking listener, bound to the IPv4 localhost address
 * 127.0.0.1, on a single implementation-defined port number. Client connections on TCP are accepted and
 * continue to be read into a large per-connection internal buffer until the connection is closed on the
 * socket. Individual TestResult objects are confined into an owning container and stored internally. Once the
 * streaming has completed, callers can execute
 *  @ref TestListenerBase::endow_test to provide ref-counted pointers to Test objects relevant to the streamed
 *  TestResult records.
 *
 * @see @ref GoogleTestLexer.l for the expected token format of the Google Test Payload
 * @see @ref GoogleTestParser.y for the expected grammar of the Google Test payload
 */
class GoogleTestListener : public TestListenerBase
{
public:
    /**
     * @brief Construct the listener by opening a new listening socket and registering callbacks.
     * @param port_number The port number of the listening socket.
     * @throws Glib::Error The socket could not be established.
     */
    explicit GoogleTestListener(guint16 port_number);

private:
    void connection_callback(const Glib::RefPtr<Gio::AsyncResult> &result) noexcept override;

    static const log4cxx::LoggerPtr logger;
    static constexpr std::size_t temp_buffer_size = 1024;

    Glib::RefPtr<Gio::SocketListener> listener = Gio::SocketListener::create();

    std::istringstream lexer_input_stream;
    GoogleTestLexer lexer{lexer_input_stream, std::cerr};
    GoogleTestParser parser;
};
```

```
} // namespace optifol

#endif // GOOGLETESTLISTENER_HPP
```

1.15.2.7.5 GoogleTestParser.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification of the exposed Google Test results protocol parser
 * @date 2025-07-14
 * @author Oliver Dixon <od641@york.ac.uk>
 * @note This file depends on build-time generated source from Flex and Bison.
 */

#ifndef GOOGLETESTPARSER_HPP
#define GOOGLETESTPARSER_HPP

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Weverything"
#include <generated/BaseGoogleTestParser.hpp> // Generated by Bison
#pragma clang diagnostic pop

#include <log4cxx/logger.h>
#include <sigc++/slot.h>
#include "../../../Exceptions/ParseError.hpp"
#include "../TestResult.hpp"

namespace optifol
{

/**
 * @class GoogleTestParser
 * @brief Provide additional functionality to the Bison-generated impl::BaseGoogleTestParser for enhanced
 * ownership semantics of produced results.
 *
 * @details In particular, the parser should obey the following sequence for all individually parsed test
 * results within their respective G-Test test cases: <ol> <li> Construct the TestResult and transfer
 * ownership to the parser in the <i>Pending</i> state with
 *          @ref add_pending_test_result. Pending results are held in @ref std::unique_ptr containers.
 *      </li>
 *      <li>
 *          Once the test fixture name is known by the innermost TestCase production, extract all pending
 * results and populate with the fixture name. Ownership of the pending TestResult is transferred back to the
 * grammar production generated member function. Use @ref steal_next_pending_result until it indicates that
 * there are no further pending results owned by the parser instance.
 *      </li>
 *      <li>
 *          As TestResult objects are populated with the test case name, use @ref push_result to invoke the
 *          parser-defined callback and transfer ownership of the TestResult to the client.
 *      </li>
 *  </ol>
 */
class GoogleTestParser : public impl::BaseGoogleTestParser
{
public:
    /**
     * @brief Instantiate a new Google Test results protocol parser, to be supplied by the given lexer
     * @param lexer The Flex-created lexer with which the parser should be acquainted
     * @param push_callback The callback to which newly parsed TestResult objects should be sent
     */
    explicit GoogleTestParser(
            GoogleTestLexer *lexer, sigc::slot<void(std::unique_ptr<TestResult> &&)> &&push_callback) :
        BaseGoogleTestParser(lexer),
        push_callback(std::move(push_callback))
    {
    }

    void error(const std::string &msg) override
    {
        Logging::get_logger({"UserTesting", "Parsing", "GoogleTest"})->error(msg);
        throw ParseError(msg, 0);
    }
```

```
    /**
     * @brief Invoke the callback to indicate a new TestResult has been produced by the parser
     * @param test_result An exclusively owning container for the constructed TestResult object
     * @note This member function must be public as it is accessed by the Bison-generated C++ grammar
     * productions code.
     */
    void push_result(std::unique_ptr<TestResult> &&test_result) const
    {
        push_callback(std::move(test_result));
    }

    /**
     * @brief Provide a new TestResult to be stored in the pending state, such that the TestResult is
     * temporarily held by the parser for further mutation before it is suitable for the push callback.
     * @param test_result The exclusively owning container for the pending TestResult object
     * @note This member function must be public as it is accessed by the Bison-generated C++ grammar
     * productions code.
     */
    void add_pending_test_result(std::unique_ptr<TestResult> &&test_result)
    {
        pending_test_results.push_back(std::move(test_result));
    }

    /**
     * @brief Extract and provide the next pending test result; this function should typically be polled until
     * there are no more pending results, as the ordering is not defined.
     * @return An exclusively owning container for the stolen TestResult, or nullptr if there are no pending
     * results.
     * @note This member function must be public as it is accessed by the Bison-generated C++ grammar
     * productions code.
     */
    std::unique_ptr<TestResult> steal_next_pending_result() noexcept
    {
        if (pending_test_results.empty())
            return nullptr;

        auto borrowed_result = std::move(pending_test_results.back());
        pending_test_results.pop_back();
        return borrowed_result;
    }

private:
    std::vector<std::unique_ptr<TestResult>> pending_test_results;

    sigc::slot<void(std::unique_ptr<TestResult> &&)> push_callback;
};

} // namespace optifol

#endif
```

1.15.2.7.6 GoogleTestParser.y

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

%require "3.7.4"
%language "C++"

%define api.parser.class {BaseGoogleTestParser}
%define api.namespace {optifol::impl}
%define api.value.type variant
%define api.prefix {GoogleTest}
%define parse.error detailed
%defines
%skeleton "lalr1.cc"
%parse-param {GoogleTestLexer* scanner}

%code requires
{
    #include <cmath>
    #include <memory>

    #include "../Exceptions/SemanticException.hpp"
    #include "../Exceptions/ParseError.hpp"
    #include "../TestResult.hpp"
    #include "../Logging.hpp"
```

```
    namespace optifol
    {
        class GoogleTestLexer;
    }
}

%code
{
    #include "GoogleTestLexer.hpp"
    #define yylex(x) scanner->lex(x)

    const log4cxx::LoggerPtr logger = optifol::Logging::get_logger({"UserTesting", "Parsing", "GoogleTest"});
}

%token ProtocolVersion
%token End
%token Passed
%token ElapsedTime
%token Name

%token <std::string> Literal

%token ProgramStart
%token ProgramEnd

%token IterationStart
%token IterationCount
%token IterationEnd

%token TestCaseStart
%token TestCaseEnd

%token TestStart
%token TestPartial
%token File
%token Line
%token Message
%token TestEnd

%type <Glib::RefPtr<PartialTestResult>> partial_test
%type <std::vector<Glib::RefPtr<PartialTestResult>>> partial_test_list

%start program_entry

%%

program_entry :
    protocol_line ProgramStart iteration_list ProgramEnd Passed Literal End
    {
        return 0;
    }
    |
    error
    {
        return -1;
    }
    ;

protocol_line :
    ProtocolVersion Literal
    {
        constexpr float supported_version = 1.0f;

        try {
            const float version = std::stof($2);
            if (fabsf(version - supported_version) >= std::numeric_limits<float>::epsilon())
                throw SemanticException("Unsupported Google Test protocol version " + $2);
        } catch (const std::logic_error& parsing_exception) {
            logger->error(parsing_exception.what());
            throw ParseError("Could not parse Google Test protocol version number \"" + $2 + '"', 0);
        } catch (const SemanticException& semantic_exception) {
            logger->error(semantic_exception.what());
            throw;
        }
    }
    ;

iteration :
    IterationStart IterationCount Literal case_list IterationEnd Passed Literal ElapsedTime Literal
    {
```

```
        // Collapse iteration lists
        if (logger->isDebugEnabled())
            logger->debug("Parsed iteration #" + $3 + ", taking " + $9 + " with an overall result of " + $7 +
                '.');
    }
    ;

iteration_list :
    %empty
    {
    }
    |
    iteration_list iteration
    {
    }
    ;

case :
    TestCaseStart Name Literal test_list TestCaseEnd Passed Literal ElapsedTime Literal
    {
        // Collapse test case sets
        if (logger->isDebugEnabled())
            logger->debug("Parsed test case \"" + $3 + "\", taking " + $9 + " with an overall result of " + $7
                + '.');

        auto& self = dynamic_cast<GoogleTestParser&>(*this);
        std::unique_ptr<TestResult> borrowed_result;

        while ((borrowed_result = self.steal_next_pending_result()) != nullptr) {
            borrowed_result->populate_test_fixture_name($3);
            self.push_result(std::move(borrowed_result));
        }
    }
    ;

case_list :
    %empty
    {
    }
    |
    case_list case
    {
    }
    ;

test :
    TestStart Name Literal partial_test_list TestEnd Passed Literal ElapsedTime Literal
    {
        try {
            const auto passed_result = std::stoi($7);
            if (passed_result != 0 && passed_result != 1)
                throw SemanticException("Invalid test result; should be '0' or '1', but received " + $7);

            /*
             * Construct the test without a test case name, and add it to a specially designated 'pending'
                 results
             * vector. Once the innermost test case has been parsed, all pending results are populated with
                 the case
             * name and moved to the 'populated' results vector.
             */
            dynamic_cast<GoogleTestParser&>(*this).add_pending_test_result(std::move(
                std::make_unique<TestResult>($3, passed_result, std::stoi($9), std::move($4))));

            if (logger->isDebugEnabled())
                logger->debug("Parsed test \"" + $3 + "\", taking " + $9 + " with an overall result of " +
                    ((passed_result) ? "PASS" : "FAIL") + '.');
        } catch (const std::logic_error& parsing_exception) {
            logger->error(parsing_exception.what());
            throw ParseError("Could not parse Google Test test result \"" + $3 + '"', 0);
        } catch (const SemanticException& semantic_exception) {
            logger->error(semantic_exception.what());
            throw;
        }
    }

test_list:
    %empty
    {
    }
    |
    test_list test
```

```
    {
    }
    ;

partial_test :
    TestPartial File Literal Line Literal Message Literal
    {
        try {
            $$ = Glib::make_refptr_for_instance(new PartialTestResult($3, std::stoi($5), $7));

            if (logger->isDebugEnabled())
                logger->debug("Parsed partial test result in file \"" + $3 +"\" at line " + $5 + '.');
        } catch (const std::logic_error& parsing_exception) {
            logger->error(parsing_exception.what());
            throw ParseError("Could not parse Google Test partial result in file \"" + $3 + "\" at line " + $5
                +
                "\": " + parsing_exception.what(), 0);
        }
    }

partial_test_list :
    %empty
    {
    }
    |
    partial_test_list partial_test
    {
        $$ = std::move($1);
        $$.push_back(std::move($2));
    }
    ;

%%

void optifol::impl::BaseGoogleTestParser::error(const std::string& msg)
{
    dynamic_cast<GoogleTestParser&>(*this).error(msg);
}
```

1.15.2.7.7 TestListenerBase.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the generic automated test socket listener
 * @author Oliver Dixon
 * @date 2025-07-27
 * @version Development
 */

#include "TestListenerBase.hpp"
#include "../../../Storage/Requirement.hpp"

namespace optifol
{

void TestListenerBase::accept_result(std::unique_ptr<TestResult> &&test_result)
{
    received_test_blob.emplace(std::move(test_result));
}

void TestListenerBase::endow_test(Test &candidate)
{
    // TestResults are keyed on the fixture and test name. If a Test demands one that is in our blob, share
    // it.
    const auto result_it = received_test_blob.find(
            std::make_pair(candidate.property_fixture().get_value(), candidate.property_name().get_value()));

    if (result_it != received_test_blob.cend())
        candidate.emplace_result(*result_it);
}

} // namespace optifol
```

1.15.2.7.8 TestListenerBase.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the generic automated test socket listener
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#ifndef TESTLISTENERBASE_HPP
#define TESTLISTENERBASE_HPP

#include <giomm/asyncresult.h>
#include <glibmm/refptr.h>
#include <memory>

#include "../Optifol.hpp"
#include "../TestResult.hpp"

namespace optifol
{

class Test;

/**
 * @class TestListenerBase
 * @brief Provides a test-framework-agnostic listener to accept network-streamed payloads describing results
 * of automated tests. Asynchronous network operations are provided by the Glib socket abstraction layers.
 * Received TestResult objects are stored by the listener in a blob and can be distributed (through shared
 * ownership) to relevant Test objects with @ref endow_test.
 */
class TestListenerBase
{
public:
    /**
     * @brief Destruct the TestListenerBase, discarding any unused TestResult objects and closing any opened
     * network state.
     */
    virtual ~TestListenerBase() = default;

    /**
     * @brief Accepts a newly formed TestResult object
     * @param test_result The owning container of the constructed TestResult
     */
    void accept_result(std::unique_ptr<TestResult> &&test_result);

    /**
     * @brief Given a Requirement with an associated Test object, determine whether a stored TestResult
     * matches the test specification of the Requirement. If it does, share the TestResult with the
     * Requirement.
     * @param candidate The Requirement to consider sharing the TestResult
     * @throws SemanticException if the Requirement refused the TestResult
     * @see Requirement::emplace_test_result
     */
    void endow_test(Test &candidate);

protected:
    /**
     * @brief Handle an Glib-asynchronously accepted connection from a client, ready to receive data.
     * @param result The result of the asynchronous socket operation.
     * @note All implementations of this member function must be noexcept as they are called from an
     * asynchronous Glib-defined context. Any exceptions thrown by composed routines should be absorbed by the
     * callback and logged using log4cxx instance for the implementation class.
     */
    virtual void connection_callback(const Glib::RefPtr<Gio::AsyncResult> &result) noexcept = 0;

private:
    SharedUnorderedSet<TestResult> received_test_blob;
};

} // namespace optifol

#endif // TESTLISTENERBASE_HPP
```

1.15.3 Modelling

1.15.3.1 ExecutionGroup.cpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the framework-agnostic execution group
 * @author Oliver Dixon
 * @date 2025-08-08
 * @version Development
 */

#include "ExecutionGroup.hpp"
#include "Test.hpp"

namespace optifol
{

bool ExecutionGroup::operator==(const TestExecutable &other) const
{
    return other.property_name().get_value() == executable.property_name().get_value();
}

bool ExecutionGroup::operator==(const ExecutionGroup &other) const
{
    return other.executable == executable;
}

Glib::ustring ExecutionGroup::get_executable_name() const
{
    return executable.property_name().get_value();
}

ExecutionGroup::ExecutionGroup(const TestExecutable &executable) :
    executable(executable)
{
}

bool ExecutionGroup::check_eligibility(const Test &test) const
{
    return *test.observe_test_executable() == executable;
}

std::size_t ExecutionGroup::hash() const noexcept
{
    return executable.hash();
}

} // namespace optifol

namespace std
{

size_t hash<optifol::ExecutionGroup>::operator()(const optifol::ExecutionGroup &object) const noexcept
{
    return object.hash();
}

size_t hash<optifol::ExecutionGroup>::operator()(const optifol::TestExecutable &executable) const noexcept
{
    return executable.hash();
}

size_t hash<optifol::ExecutionGroup>::operator()(
        const unique_ptr<optifol::ExecutionGroup> &unique_hashable) const noexcept
{
    return unique_hashable->hash();
}

} // namespace std
```

1.15.3.2 ExecutionGroup.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the framework-agnostic execution group
 * @author Oliver Dixon
 * @date 2025-08-08
 * @version Development
 */

#ifndef EXECUTIONGROUP_HPP
#define EXECUTIONGROUP_HPP

#include <glibmm/refptr.h>
#include <glibmm/ustring.h>
#include <sigc++/functors/slot.h>
#include <unordered_set>

#include "../../IHashable.hpp"

namespace optifol
{

class TestExecutable;
class Test;

class ExecutionGroup : public IHashable
{
public:
    [[nodiscard]] std::size_t hash() const noexcept override;

    virtual void add_test(Glib::RefPtr<Test> new_test) = 0;

    virtual void remove_test(const Glib::RefPtr<Test> &target_test) = 0;

    virtual void run() = 0;

    [[nodiscard]] virtual std::size_t is_empty() const noexcept = 0;

    [[nodiscard]] bool operator==(const TestExecutable &other) const;

    [[nodiscard]] bool operator==(const ExecutionGroup &other) const;

    [[nodiscard]] Glib::ustring get_executable_name() const;

protected:
    explicit ExecutionGroup(const TestExecutable &executable);

    [[nodiscard]] bool check_eligibility(const Test &test) const;

private:
    const TestExecutable &executable;
};

} // namespace optifol

template<>
struct std::hash<optifol::ExecutionGroup>
{
    using is_transparent = void;

    std::size_t operator()(const optifol::ExecutionGroup &object) const noexcept;

    std::size_t operator()(const optifol::TestExecutable &executable) const noexcept;

    std::size_t operator()(const unique_ptr<optifol::ExecutionGroup> &unique_hashable) const noexcept;
};


#endif // EXECUTIONGROUP_HPP
```

1.15.3.3 GoogleExecutionGroup.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
```

```
 */

/**
 * @file
 * @brief Class implementation for the Google Test execution group
 * @author Oliver Dixon
 * @date 2025-08-08
 * @version Development
 */

#include "GoogleExecutionGroup.hpp"

#include "../../Exceptions/SemanticException.hpp"
#include "GoogleTestListener.hpp"
#include "Test.hpp"

namespace optifol
{

guint16 GoogleExecutionGroup::port_number = GoogleExecutionGroup::minimum_port_number;

GoogleExecutionGroup::GoogleExecutionGroup(Glib::RefPtr<Test> initial_test) :
    ExecutionGroup(*initial_test->observe_test_executable())
{
    GoogleExecutionGroup::add_test(std::move(initial_test));
}

void GoogleExecutionGroup::run()
{
    if (is_empty())
        return;

    if (cache_ok == false)
        invalidate_cache();

    listener = std::make_unique<GoogleTestListener>(port_number);
    executor = std::make_unique<ProcessExecutor>("",
            std::vector<std::string>{get_executable_name(), "--gtest_filter=" + filter_line_cache,
                    "--gtest_stream_result_to=127.0.0.1:" + std::to_string(port_number)},
            std::vector<std::string>{}, sigc::mem_fun(*this, &GoogleExecutionGroup::distribute_results));

    if (++port_number == maximum_port_number)
        port_number = minimum_port_number;
}

void GoogleExecutionGroup::add_test(Glib::RefPtr<Test> new_test)
{
    if (check_eligibility(*new_test) == false)
        throw SemanticException("The Test does not match to the GoogleExecutionGroup.");

    tests.push_front(std::move(new_test));
    cache_ok = false;
}

void GoogleExecutionGroup::remove_test(const Glib::RefPtr<Test> &target_test)
{
    tests.remove(target_test);
    cache_ok = false;
}

std::size_t GoogleExecutionGroup::is_empty() const noexcept
{
    return tests.empty();
}

void GoogleExecutionGroup::distribute_results(const int exit_code) const
{
    std::ignore = exit_code;
    for (auto &test: tests)
        listener->endow_test(*test);
}

void GoogleExecutionGroup::invalidate_cache()
{
    filter_line_cache.clear();

    for (const auto &test: tests)
        filter_line_cache +=
                test->property_fixture().get_value() + '.' + test->property_name().get_value() + ':';

    cache_ok = true;
```

```
}

} // namespace optifol
```

1.15.3.4 GoogleExecutionGroup.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Google Test execution group
 * @author Oliver Dixon
 * @date 2025-08-08
 * @version Development
 */

#ifndef GOOGLEEXECUTIONGROUP_HPP
#define GOOGLEEXECUTIONGROUP_HPP

#include <forward_list>

#include "../../GUI/ProcessExecutor.hpp"
#include "ExecutionGroup.hpp"
#include "Test.hpp"
#include "TestListenerBase.hpp"

namespace optifol
{

class GoogleExecutionGroup : public ExecutionGroup
{
public:
    explicit GoogleExecutionGroup(Glib::RefPtr<Test> initial_test);

    void run() override;

    void add_test(Glib::RefPtr<Test> new_test) override;

    void remove_test(const Glib::RefPtr<Test> &target_test) override;

    /**
     * @brief Queries the empty state of the Test list.
     * @return Does the ExecutionGroup contain any Test objects?
     */
    [[nodiscard]] std::size_t is_empty() const noexcept override;

private:
    /**
     * @brief Handle the sub-process exit by distributing results from the listener pool to the Requirement
     * objects.
     * @param exit_code Exit code from the Google Test sub-process.
     */
    void distribute_results(int exit_code) const;

    /**
     * @brief Force a rebuild of the @ref filter_line_cache string from the @ref tests content.
     */
    void invalidate_cache();

    static constexpr char pattern_separator = ':';
    static constexpr char component_separator = '.';

    static constexpr guint16 minimum_port_number = 1024; // First unprivileged port number under Linux.
    static constexpr guint16 maximum_port_number = std::numeric_limits<guint16>::max();
    static guint16 port_number;

    std::unique_ptr<ProcessExecutor> executor;
    std::unique_ptr<TestListenerBase> listener;

    std::forward_list<Glib::RefPtr<Test>> tests;

    /**
     * @brief The Google Test filter specification for the loaded @ref tests. The Google Test format for a
     * single fixture-test pair is <code>fixture.test:</code>. The colon is a delimiter and may be trailing.
     */
    std::string filter_line_cache;
```

```
    /**
     * @brief Is @ref filter_line_cache a representative Google Test filter string for the detained @ref
     * tests?
     */
    bool cache_ok = true;
};

} // namespace optifol

#endif // GOOGLEEXECUTIONGROUP_HPP
```

1.15.3.5 ITestModelNode.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the ITestModelNode interface
 * @author Oliver Dixon
 * @date 2025-08-02
 * @version Development
 */

#ifndef TESTMODELNODE_HPP
#define TESTMODELNODE_HPP

#include <gtkmm/treelistmodel.h>

namespace optifol
{

/**
 * @class ITestModelNode
 * @brief Provides an interface for participation in the hierarchy of runnable software-level Test objects.
 * Such nodes expose two retrieval functions to get child Gtk::TreeListModel objects for the Test and
 * TestResult trees. These can be accessed through a Glib-friendly static context with @ref
 * ITestModelNode::get_given_tests_tree and
 *  @ref ITestModelNode::get_given_results_tree.
 */
class ITestModelNode
{
public:
    /**
     * @brief Destruct the ITestModelNode.
     */
    virtual ~ITestModelNode() = default;

    /**
     * @brief Statically retrieve the Test object tree from the given ITestModelNode detained by a
     * Glib::ObjectBase.
     * @param node The ITestModelNode, wrapped in a Glib::ObjectBase.
     * @return The Test object tree, or an empty Glib::RefPtr for a leaf node or node of incorrect type.
     */
    static Glib::RefPtr<Gtk::TreeListModel> get_given_tests_tree(
            const Glib::RefPtr<Glib::ObjectBase> &node) noexcept
    {
        const auto typed_node = dynamic_cast<ITestModelNode *>(node.get());
        if (typed_node == nullptr)
            return nullptr;

        return typed_node->get_tests_tree();
    }

    /**
     * @brief Statically retrieve the TestResult object tree from the given ITestModelNode detained by a
     *  Glib::ObjectBase.
     * @param node The ITestModelNode, wrapped in a Glib::ObjectBase.
     * @return The TestResult object tree, or an empty Glib::RefPtr for a leaf node or node of incorrect type.
     */
    static Glib::RefPtr<Gtk::TreeListModel> get_given_results_tree(
            const Glib::RefPtr<Glib::ObjectBase> &node) noexcept
    {
        const auto typed_node = dynamic_cast<ITestModelNode *>(node.get());
        if (typed_node == nullptr)
            return nullptr;
```

```
        return typed_node->get_results_tree();
    }

private:
    /**
     * @brief Retrieve the Gtk::TreeListModel containing the child Test objects.
     * @return The tree, or an empty Glib::RefPtr for a leaf node.
     */
    [[nodiscard]] virtual Glib::RefPtr<Gtk::TreeListModel> get_tests_tree() const noexcept = 0;

    /**
     * @brief Retrieve the Gtk::TreeListModel containing the results of the child Test objects.
     * @return The tree, or an empty Glib::RefPtr for a leaf node.
     */
    [[nodiscard]] virtual Glib::RefPtr<Gtk::TreeListModel> get_results_tree() const noexcept = 0;
};

} // namespace optifol

#endif // TESTMODELNODE_HPP
```

1.15.3.6 Test.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Test-level storage object
 * @author Oliver Dixon
 * @date 2025-07-17
 * @version Development
 */

#include "Test.hpp"
#include "../../Exceptions/SemanticException.hpp"
#include "../Discovery/DiscoveryTestFixture.hpp"

namespace optifol
{

Test::Test(std::shared_ptr<TestSpecificationEntry> template_specification) :
    Glib::ObjectBase("Test"),
    target_executable_name(*this, "Test-target-executable-name"),
    fixture(*this, "Test-test-fixture"),
    result(*this, "Test-result")
{
    instantiate_from_specification(std::move(template_specification));
}

Test::Test(std::shared_ptr<TestSpecificationEntry> template_specification, BaseObjectType *cobject,
        const Glib::RefPtr<Gtk::Builder> &builder) :
    Glib::ObjectBase("Test"),
    StorageObjectBase(cobject, builder),
    target_executable_name(*this, "Test-target-executable-name"),
    fixture(*this, "Test-test-fixture"),
    result(*this, "Test-result")
{
    instantiate_from_specification(std::move(template_specification));
}

bool Test::operator==(const Test &other) const
{
    return property_target_executable_name().get_value() ==
            other.property_target_executable_name().get_value() &&
            property_fixture().get_value() == other.property_fixture().get_value() &&
            property_name().get_value() == other.property_name().get_value();
}

Glib::RefPtr<Gtk::TreeListModel> Test::get_tests_tree() const noexcept
{
    return nullptr;
}

Glib::RefPtr<Gtk::TreeListModel> Test::get_results_tree() const noexcept
{
```

```
    if (result.get_value() == nullptr)
        return {};

    return result.get_value()->get_results_tree();
}

void Test::emplace_result(const std::shared_ptr<TestResult> &test_result)
{
    const auto &given_fixture_name = test_result->copy_fixture_name();
    const auto &expected_fixture_name = fixture.get_value();

    if (given_fixture_name != expected_fixture_name)
        throw SemanticException("Incoming test result was from a different fixture: \"" + given_fixture_name +
                "\", but needed \"" + expected_fixture_name + "\".");

    if (test_result->copy_test_name() != property_name().get_value())
        throw SemanticException("Incoming test result was from a different test: \"" +
                test_result->copy_test_name() + "\", but needed \"" + property_name().get_value() + "\".");

    result.set_value(test_result);
}

void Test::accept_test_executable(std::shared_ptr<TestExecutable> shared_exe)
{
    target_executable = std::move(shared_exe);

    if (target_executable == nullptr)
        target_executable_name.set_value("");
    else
        target_executable_name.set_value(target_executable->property_name().get_value());
}

const TestExecutable *Test::observe_test_executable() const noexcept
{
    return target_executable.get();
}

Glib::PropertyProxy_ReadOnly<Glib::ustring> Test::property_target_executable_name() const
{
    return target_executable_name.get_proxy();
}

Glib::PropertyProxy<Glib::ustring> Test::property_fixture()
{
    return fixture.get_proxy();
}

Glib::PropertyProxy<std::shared_ptr<TestResult>> Test::property_result()
{
    return result.get_proxy();
}

Glib::PropertyProxy_ReadOnly<Glib::ustring> Test::property_fixture() const
{
    return fixture.get_proxy();
}

Glib::PropertyProxy_ReadOnly<std::shared_ptr<TestResult>> Test::property_result() const
{
    return result.get_proxy();
}

void Test::instantiate_from_specification(std::shared_ptr<TestSpecificationEntry> spec)
{
    accept_test_executable(std::make_shared<TestExecutable>(
            spec->property_executable().get_value()->property_name().get_value()));
    property_fixture().set_value(spec->property_fixture().get_value()->property_name().get_value());
    property_name().set_value(spec->property_name().get_value());

    spec->property_name().signal_changed().connect(
            [this, spec] { property_name().set_value(spec->property_name().get_value()); });

    spec->property_fixture().signal_changed().connect(
            [this, spec]
            {
                property_fixture().set_value(
                        spec->property_fixture().get_value()->property_name().get_value());
            });
}

} // namespace optifol
```

1.15.3.7 TestGroup.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the TestGroup grouping GLib object
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#include <cassert>
#include <gtkmm/label.h>
#include <gtkmm/listitem.h>

#include "../../Exceptions/SemanticException.hpp"
#include "../Logging.hpp"
#include "GoogleExecutionGroup.hpp"
#include "TestGroup.hpp"

namespace optifol
{

const log4cxx::LoggerPtr TestGroup::testgroup_logger =
        Logging::get_logger({"GUI", "StorageControl", "TestGroup"});

TestGroup::TestGroup(const Glib::ustring &name) :
    Glib::ObjectBase("TestGroup"),
    ObjectGroup(sigc::mem_fun(*this, &TestGroup::handle_requirement_model_change))
{
    property_name().set_value(name);
}

TestGroup::TestGroup(
        const Glib::ustring &name, BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder) :
    Glib::ObjectBase("TestGroup"),
    StorageObjectBase(cobject, builder),
    ObjectGroup(sigc::mem_fun(*this, &TestGroup::handle_requirement_model_change))
{
    property_name().set_value(name);
}

bool TestGroup::operator==(const TestGroup &other) const noexcept
{
    return property_name().get_value() == other.property_name().get_value();
}

Glib::RefPtr<Gtk::TreeListModel> TestGroup::get_tests_tree() const noexcept
{
    return tests_tree;
}

Glib::RefPtr<Gtk::TreeListModel> TestGroup::get_results_tree() const noexcept
{
    return results_tree;
}

decltype(TestGroup::execution_groups)::const_iterator TestGroup::begin_execution_groups() const noexcept
{
    return execution_groups.cbegin();
}

decltype(TestGroup::execution_groups)::const_iterator TestGroup::end_execution_groups() const noexcept
{
    return execution_groups.cend();
}

void TestGroup::bind_name_to_label(const Glib::RefPtr<Gtk::ListItem> &item) noexcept
{
    const auto target_label = dynamic_cast<Gtk::Label *>(item->get_child());
    const auto typed_group = dynamic_cast<const TestGroup *>(item->get_item().get());
    if (target_label == nullptr || typed_group == nullptr)
        return;

    target_label->set_text(typed_group->property_name().get_value());
}
```

```
void TestGroup::handle_requirement_model_change(
        const guint initial_index, const guint removed_count, const guint added_count) noexcept
{
    testgroup_logger->debug("Handling requirements change: " + std::to_string(added_count) +
            " additions and " + std::to_string(removed_count) + " deletions at position " +
            std::to_string(initial_index) + '.');

    handle_test_deletions(initial_index, removed_count);
    handle_test_additions(initial_index, added_count);
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from 'handle_test_deletions'
// callback.
void TestGroup::deregister_test_results(Glib::RefPtr<Test> test)
{
    assert(test != nullptr);
    const auto callback_it = registered_callbacks.find(test);

    if (callback_it == registered_callbacks.cend())
        testgroup_logger->error("Removed Test \"" + test->property_name().get_value() +
                "\" did not have a registered callback in Test Group \"" + property_name().get_value() +
                "\".");
    else {
        callback_it->second.disconnect();
        registered_callbacks.erase(callback_it);
        testgroup_logger->info("Disconnected results signal handler for Test \"" +
                test->property_name().get_value() + "\" in Test Group \"" + property_name().get_value() +
                "\".");
    }

    if (results_model.delete_object(std::move(test)))
        testgroup_logger->info(
                "Removed Test \"" + test->property_name().get_value() + "\" from the failed results model.");
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from 'handle_test_deletions'
// callback.
void TestGroup::deregister_test_executable(const Glib::RefPtr<Test> &test)
{
    assert(test != nullptr);

    /*
     * Locate the ExecutionGroup that should contain the Test, according to its TargetExecutable, and remove
     * it from the ExecutionGroup. If there is no suitable ExecutionGroup, then that is considered an error as
     * all tests have associated TargetExecutables, and should have been allocated appropriately upon being
     * registered with the TestGroup.
     */
    const auto exe_group_it = execution_groups.find(*test->observe_test_executable());
    if (exe_group_it == execution_groups.cend())
        testgroup_logger->error("Test \"" + test->property_name().get_value() +
                "\" was not a member of any Execution Groups in \"" + property_name().get_value() + "\".");
    else {
        (*exe_group_it)->remove_test(test);
        testgroup_logger->info("Removed Test \"" + test->property_name().get_value() +
                "\" from Execution Group for \"" + (*exe_group_it)->get_executable_name() + "\".");
    }

    // If the execution group is now empty, remove it.
    if ((*exe_group_it)->is_empty()) {
        testgroup_logger->info(
                "Removed empty Execution Group for \"" + (*exe_group_it)->get_executable_name() + "\".");
        execution_groups.erase(exe_group_it);
    }
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from 'handle_test_additions'
// callback.
void TestGroup::register_test_results(const Glib::RefPtr<Test> &test)
{
    assert(test != nullptr);

    /*
     * Register a results callback so we're informed each time a new TestResult is emplaced into the Test.
     * This needs to be recorded so it can be disconnected when the test is deleted. Note that TestGroups do
     * not permit duplicates, so a regular hash table, keyed on the Test, is acceptable.
     */
    registered_callbacks.emplace(test,
            test->property_result().signal_changed().connect(
                    sigc::bind(sigc::mem_fun(*this, &TestGroup::handle_incoming_result), test)));
```

```
    testgroup_logger->info("Connected results signal handler for Test \"" +
            test->property_name().get_value() + "\" in Test Group \"" + property_name().get_value() + "\".");
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive: called from 'handle_test_additions'
// callback.
void TestGroup::register_test_executable(const Glib::RefPtr<Test> &test)
{
    assert(test != nullptr);

    try {
        const auto group_it = execution_groups.find(*test->observe_test_executable());
        if (group_it == execution_groups.cend()) {
            // The first test we've seen using this executable. Create a new execution group.
            execution_groups.emplace(std::make_unique<GoogleExecutionGroup>(test));
            testgroup_logger->info("Created new Execution Group for executable \"" +
                    test->observe_test_executable()->property_name().get_value() + "\".");
        } else
            // We've seen this executable before. Add it to the existing execution group.
            group_it->get()->add_test(test);

        testgroup_logger->info("Added Test \"" + test->property_name().get_value() +
                "\" to Execution Group for \"" +
                test->observe_test_executable()->property_name().get_value() + "\".");

    } catch (const SemanticException &semantic_exception) {
        testgroup_logger->error(
                "Could not propagate Test addition for Test \"" + test->property_name().get_value() + "\".");
        testgroup_logger->error(semantic_exception.what());
    }
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive. Called from 'handle_object_change'.
void TestGroup::handle_test_deletions(const guint initial_index, const guint removed_count) noexcept
{
    for (guint remove_count_i = 0; remove_count_i < removed_count; ++remove_count_i)
        // Get the deletion record, keyed by the old index, to recover the deleted Requirement and Test.
        try {
            const auto deleted_it = steal_deleted_object(initial_index + remove_count_i);
            const auto deleted_tests = deleted_it->get_tests();
            const auto deleted_test_count = deleted_tests->get_n_items();

            if (deleted_test_count == 0)
                continue; // Nothing to do...

            for (guint deleted_test_index = 0; deleted_test_index < deleted_test_count;
                    ++deleted_test_index) {
                const auto test = deleted_tests->get_item(deleted_test_index);
                deregister_test_executable(test);
                deregister_test_results(test);
            }
        } catch (const std::runtime_error &global_error) {
            testgroup_logger->error("Could not propagate any deletions of " + std::to_string(removed_count) +
                    " from index " + std::to_string(initial_index) + " for Test Group " +
                    property_name().get_value() + "\".");
            testgroup_logger->error(global_error.what());
        }
}

// ReSharper disable once CppDFAUnreachableFunctionCall - False positive. Called from 'handle_object_change'.
void TestGroup::handle_test_additions(const guint initial_index, const guint added_count) noexcept
{
    for (guint added_count_i = 0; added_count_i < added_count; ++added_count_i)
        try {
            // Get the Tests from the incoming Requirement.
            const auto &tests = get_object_by_index(added_count_i + initial_index)->get_tests();
            const auto added_test_count = tests->get_n_items();

            // For each Test, register a results handler and allocate to the suitable execution group.
            for (guint added_test_index = 0; added_test_index < added_test_count; ++added_test_index) {
                const auto test = tests->get_item(added_test_index);
                register_test_results(test);
                register_test_executable(test);
            }

        } catch (const std::runtime_error &global_error) {
            testgroup_logger->error("Could not propagate any additions of " + std::to_string(added_count) +
                    " from index " + std::to_string(initial_index) + " for Test Group " +
                    property_name().get_value() + "\".");
            testgroup_logger->error(global_error.what());
        }
```

```
}

void TestGroup::handle_incoming_result(const std::shared_ptr<Test> &owning_test) noexcept
{
    if (owning_test == nullptr) {
        testgroup_logger->error("Test Group \"" + property_name().get_value() +
                "\" was informed on result of non-existent Test.");
        return;
    }

    const auto test_already_exists = results_model.contains_exact(owning_test.get());
    const auto result = owning_test->property_result().get_value().get();

    if (result == nullptr || result->get_results_tree()->get_n_items() == 0) {
        if (test_already_exists) {
            results_model.delete_object(owning_test);
            testgroup_logger->info("Removed Test \"" + owning_test->property_name().get_value() +
                    "\" from the failed model for the Test Group \"" + property_name().get_value() + "\".");
        }

        return;
    }

    // The test has partial results, so add it to the results model if it doesn't already exist.
    if (test_already_exists == false) {
        try {
            results_model.insert_object(owning_test);
            testgroup_logger->info("Added Test \"" + owning_test->property_name().get_value() +
                    "\" to the failed model for the Test Group \"" + property_name().get_value() + "\".");
        } catch (const std::runtime_error &error) {
            testgroup_logger->error("Could not add Test \"" + owning_test->property_name().get_value() +
                    "\" to the failed model for the Test Group \"" + property_name().get_value() + "\".");
            testgroup_logger->error(error.what());
        }
    } else
        testgroup_logger->debug("Received valid failure result for Test \"" +
                owning_test->property_name().get_value() +
                "\", but it already exists in the model for Test Group \"" + property_name().get_value() +
                "\".");
}

} // namespace optifol
```

1.15.3.8 TestGroup.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the TestGroup grouping GLib object
 * @author Oliver Dixon
 * @date 2025-07-20
 * @version Development
 */

#ifndef TESTGROUP_HPP
#define TESTGROUP_HPP

#include "../../Storage/ObjectGroup.hpp"
#include "../../Storage/Requirement.hpp"
#include "ExecutionGroup.hpp"

namespace optifol
{

/**
 * @class TestGroup
 * @brief A TestGroup is a user-populated structure that groups Test objects.
 * @details
 *  Internally, a TestGroup maintains a number of models that must be synchronised. Most of the implementation
 * of TestGroup is focused on keeping these models synchronised through a series of callbacks. There are three
 * models of note: <ol> <li> <b>The Requirement model.</b> Provided by the ObjectGroup<Requirement> base
 * class, the TestGroup groups Requirement objects endowed with Test objects. This may seem counter-intuitive,
 * but it is desired behaviour. It means that Test objects must be added via their constituent Requirement
 * parents, which allows the TestGroup to directly expose Test instances grouped according to the
 * corresponding Requirement. As the ObjectGroup CRTP base class is being used, Requirement objects are
```

```
 * transparently held in both a linear model (iteration) and a hashed model (lookup) that are invariantly
 * synchronised.
 *      </li>
 *      <li>
 *          <b>The ExecutionGroup model.</b> Provided by an implementation detail, ExecutionGroup objects
 * aggregate Test objects (irrespective of Requirement) according to their associated TestExecutable.
 * Therefore, in addition to a model centred on Requirement, there is a model centred on TestExecutable.
 * ExecutionGroup objects can be accessed via the iterator-based member functions @ref begin_execution_groups
 * and
 *          @ref end_execution_groups.
 *      </li>
 *      <li>
 *          <b>The TestResult and PartialTestResult model.</b> Provided by ObjectGroup<Test>, the TestGroup
 * groups TestResult and PartialTestResult objects based on the Test to which they refer, irrespective of
 *          Requirement or TestExecutable. This enables a second 'view' of the TestGroup, accessible via the
 *          ITestModelNode member functions, displaying TestResult information without regard to the
 * Requirement structure.
 *      </li>
 *  </ol>
 */
class TestGroup : public StorageObjectBase,
                  public ObjectGroup<Requirement>,
                  public ITestModelNode
{
    UniqueUnorderedSet<ExecutionGroup> execution_groups;

public:
    /**
     * @brief Constructs a new TestGroup with the given name
     * @param name The initial name of the TestGroup
     */
    explicit TestGroup(const Glib::ustring &name);

    /**
     * @brief Constructs a new TestGroup with the given name
     * @param name The initial name of the TestGroup
     * @param cobject The base C GTK object
     * @param builder The source GTK builder object
     */
    TestGroup(const Glib::ustring &name, BaseObjectType *cobject, const Glib::RefPtr<Gtk::Builder> &builder);

    /**
     * @brief Determines equality between two TestGroup objects based on their name
     * @param other The TestGroup with which equality should be tested.
     * @return Is the object equivalent to the given TestGroup?
     */
    [[nodiscard]] bool operator==(const TestGroup &other) const noexcept;

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_tests_tree() const noexcept override;

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_results_tree() const noexcept override;

    /**
     * @brief Get the constant iterator to the beginning of the ordered execution groups.
     * @return Constant iterator to the beginning of the execution groups.
     */
    decltype(execution_groups)::const_iterator begin_execution_groups() const noexcept;

    /**
     * @brief Get the constant iterator to the one-past-end of the ordered execution groups.
     * @return Constant iterator to the one-past-end of the execution groups.
     */
    decltype(execution_groups)::const_iterator end_execution_groups() const noexcept;

    /**
     * @brief Bind the name of the TestGroup to a Gtk::Label.
     * @param item A Gtk::ListItem with a Gtk::Label child and TestGroup item.
     */
    static void bind_name_to_label(const Glib::RefPtr<Gtk::ListItem> &item) noexcept;

private:
    /**
     * @brief Handles a change to the underlying object model by registering and/or de-registering Test
     * objects from Requirement objects.
     * @param initial_index The initial index in the list model from which changes were made.
     * @param removed_count The number of removed items at the initial index.
     * @param added_count The number of inserted items at the initial index.
     * @see handle_test_deletions for the deletion handler
     * @see handle_test_additions for the addition handler
     */
    void handle_requirement_model_change(
```

```
            guint initial_index, guint removed_count, guint added_count) noexcept;

    /**
     * @brief Remove the TestResult model entries for the given Test.
     * @param test The slated Test associated with the TestResult model entries to de-register.
     * @see @ref results_model for the model.
     * @pre The given Test container is non-null.
     */
    void deregister_test_results(Glib::RefPtr<Test> test);

    /**
     * @brief Remove the given Test from the corresponding ExecutionGroup within the group model. If the
     * removal renders the corresponding ExecutionGroup empty, it is removed from the model entirely.
     * @param test The Test to remove from the ExecutionGroup.
     * @see @ref execution_groups for the model.
     * @pre The given Test container is non-null.
     */
    void deregister_test_executable(const Glib::RefPtr<Test> &test);

    /**
     * @brief Register a new callback to listen for changes to the TestResult objects assigned to the given
     * Test, and synchronise the changes in our local @ref results_model.
     * @param test The Test whose results to watch.
     * @pre The given Test container is non-null.
     */
    void register_test_results(const Glib::RefPtr<Test> &test);

    /**
     * @brief Allocate the incoming Test to the suitable ExecutionGroup, determined by the Test's target
     * executable property. A new ExecutionGroup is created if one does not already exist.
     * @param test The Test to register in the execution model.
     * @throws std::runtime_error if the Test could not be added.
     * @pre The given Test container is non-null.
     */
    void register_test_executable(const Glib::RefPtr<Test> &test);

    /**
     * @brief Handle Test deletions by deregistering watchers for TestResult changes, and removing from the
     *  ExecutionGroup model.
     * @param initial_index The initial index of the deleted Test objects; see @ref
     * ObjectGroup::steal_deleted_object.
     * @param removed_count The number of removed Test objects from the initial index.
     * @see deregister_test_executable for removal from ExecutionGroup model.
     * @see deregister_test_results for removal from TestResult model.
     */
    void handle_test_deletions(guint initial_index, guint removed_count) noexcept;

    /**
     * @brief Handle Tets additions by registering watchers for TestResult changes, and inserting into the
     *  ExecutionGroup model.
     * @param initial_index The initial index of the inserted Test objects; see @ref
     * ObjectGroup::get_object_by_index.
     * @param added_count The number of inserted Test objects from the initial index.
     * @see register_test_results for insertion into TestResult model.
     * @see register_test_executable for insertion into ExecutionGroup model.
     */
    void handle_test_additions(guint initial_index, guint added_count) noexcept;

    /**
     * @brief Handle a new TestResult object for the given Test.
     * @param owning_test The Test containing the new result.
     */
    void handle_incoming_result(const std::shared_ptr<Test> &owning_test) noexcept;

    static const log4cxx::LoggerPtr testgroup_logger;

    Glib::RefPtr<Gtk::TreeListModel> tests_tree =
            Gtk::TreeListModel::create(get_model(), &ITestModelNode::get_given_tests_tree, true);

    ObjectGroup<Test> results_model;

    SharedUnorderedMap<Test, sigc::connection> registered_callbacks;

    Glib::RefPtr<Gtk::TreeListModel> results_tree = Gtk::TreeListModel::create(
            results_model.get_model(), &ITestModelNode::get_given_results_tree, true);
};

} // namespace optifol

#endif // TESTGROUP_HPP
```

1.15.3.9 Test.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Test-level storage object
 * @author Oliver Dixon
 * @date 2025-07-17
 * @version Development
 */

#ifndef TEST_HPP
#define TEST_HPP

#include "../../Storage/StorageObjectBase.hpp"
#include "../Discovery/TestSpecificationEntry.hpp"
#include "../Execution/TestExecutable.hpp"
#include "../Execution/TestResult.hpp"
#include "ITestModelNode.hpp"

namespace optifol
{

/**
 * @class Test
 * @brief The Test storage object denotes a single unit test to be executed against a testable target
 * executable.
 */
class Test : public StorageObjectBase,
             public ITestModelNode
{
public:
    explicit Test(std::shared_ptr<TestSpecificationEntry> template_specification);

    Test(std::shared_ptr<TestSpecificationEntry> template_specification, BaseObjectType *cobject,
         const Glib::RefPtr<Gtk::Builder> &builder);

    [[nodiscard]] bool operator==(const Test &other) const;

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_tests_tree() const noexcept override;

    [[nodiscard]] Glib::RefPtr<Gtk::TreeListModel> get_results_tree() const noexcept override;

    /**
     * @brief Accept a shared TestResult object to indicate the last-known result of the Test.
     * @param test_result The TestResult to share.
     * @throws SemanticException if the TestResult was not appropriate or relevant to the Test.
     */
    void emplace_result(const std::shared_ptr<TestResult> &test_result);

    void accept_test_executable(std::shared_ptr<TestExecutable> shared_exe);

    [[nodiscard]] const TestExecutable *observe_test_executable() const noexcept;

    [[nodiscard]] Glib::PropertyProxy<Glib::ustring> property_fixture();

    [[nodiscard]] Glib::PropertyProxy<std::shared_ptr<TestResult>> property_result();

    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::ustring> property_target_executable_name() const;

    [[nodiscard]] Glib::PropertyProxy_ReadOnly<Glib::ustring> property_fixture() const;

    [[nodiscard]] Glib::PropertyProxy_ReadOnly<std::shared_ptr<TestResult>> property_result() const;

private:
    void instantiate_from_specification(std::shared_ptr<TestSpecificationEntry> spec);

    std::shared_ptr<TestExecutable> target_executable;
    Glib::Property<Glib::ustring> target_executable_name;
    Glib::Property<Glib::ustring> fixture;
    Glib::Property<std::shared_ptr<TestResult>> result;
};

} // namespace optifol

#endif // TEST_HPP
```

1.16 Visitors

1.16.1 MutableTargets

1.16.1.1 RepositoryBuildingVisitor.cpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Symbol Repository-Building Visitor
 * @author Oliver Dixon
 * @date 2025-06-22
 * @version Development
 */

#include "RepositoryBuildingVisitor.hpp"

#include <cassert>

#include "../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../../IR/MutableVariants/Sentences/MutablePredicate.hpp"
#include "../../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"
#include "../../IR/MutableVariants/Terms/MutableFunction.hpp"
#include "../../IR/MutableVariants/Terms/MutableSkolemFunction.hpp"
#include "../../IR/MutableVariants/Terms/MutableVariable.hpp"
#include "../../IR/Sentences/Literal.hpp"
#include "../../IR/Sentences/SentenceRoot.hpp"
#include "../../IR/Terms/Function.hpp"
#include "../../IR/Terms/SkolemFunction.hpp"
#include "../../IR/Terms/Variable.hpp"

namespace optifol
{

const char *RepositoryBuildingVisitor::visitor_name = "Repository-Building Visitor";

RepositoryBuildingVisitor::RepositoryBuildingVisitor(std::shared_ptr<SymbolRepository> symbol_repository) :
    symbol_repository(std::move(symbol_repository))
{
}

std::string_view RepositoryBuildingVisitor::get_visitor_name()
{
    return visitor_name;
}

const IProcessedSentence *RepositoryBuildingVisitor::visit(const MutableQuantified &node)
{
    std::ignore = node;
    assert(false);
    // ReSharper disable once CppDFAUnreachableCode - Not unreachable if assertions disabled.
    return nullptr;
}

const BinaryConnected *RepositoryBuildingVisitor::visit(MutableBinaryConnected &node)
{
    const auto operator_type = node.get_operator_type();
    assert(operator_type == BinaryOperatorTypes::Conjunction ||
           operator_type == BinaryOperatorTypes::Disjunction);

    /*
     * The "manage clause" flag indicates that this function is responsible for committing and resetting the
     * working clause member function, and that its callees are exclusively allowed to insert literals into
     * the working clause. Note that this should be set for any conjunctive node, and if an existing non-empty
     * working clause is present, it may be safely assumed to be fully populated and committed to the sentence
     * root.
     */
    const bool managed_clause = operator_type == BinaryOperatorTypes::Conjunction;

    // Create a fresh clause for the LHS operand, committing a previously populated clause if necessary.
    if (managed_clause && !working_clause.empty())
        commit_working_clause();

    const auto bound_lhs = node.take_lhs_operand();
    const auto repo_lhs = bound_lhs->accept(*this);
```

```
    /*
     * If the LHS recursion produced any literals under disjunction, commit the set to the sentence root, and
     * create a fresh clause for the RHS operand.
     */
    if (managed_clause && !working_clause.empty())
        commit_working_clause();

    const auto bound_rhs = node.take_rhs_operand();
    const auto repo_rhs = bound_rhs->accept(*this);

    // Likewise, commit any disjunctive literals produced by the RHS recursion to the sentence root.
    if (managed_clause && !working_clause.empty())
        commit_working_clause();

    return symbol_repository->add_symbol<BinaryConnected>(
            std::make_unique<BinaryConnected>(node.get_operator_type(), repo_lhs, repo_rhs));
}

const Literal *RepositoryBuildingVisitor::visit(MutablePredicate &node)
{
    const auto &owned_terms = node.observe_arguments();
    std::vector<const IProcessedTerm *> processed_terms;
    processed_terms.reserve(owned_terms.size());

    for (const auto &term: owned_terms)
        processed_terms.push_back(term->accept(*this));

    // Create the Literal symbol and append to the working clause.
    const auto new_symbol = symbol_repository->add_symbol<Literal>(std::make_unique<Literal>(
            std::string(node.get_name()), std::move(processed_terms), !node.is_negative_polarity()));
    working_clause.add_literal(new_symbol);
    return new_symbol;
}

void RepositoryBuildingVisitor::visit(MutableSentenceRoot &node)
{
    assert(root == nullptr);
    root = std::make_unique<SentenceRoot>();
    const auto sentence = node.take_sentence();
    sentence->accept(*this);

    symbol_repository->increment_sentence_count();

    if (!working_clause.empty())
        commit_working_clause();
}

std::unique_ptr<SentenceRoot> RepositoryBuildingVisitor::take_last_root() noexcept
{
    assert(root != nullptr);
    return std::move(root);
}

void RepositoryBuildingVisitor::commit_working_clause()
{
    root->add_clause(std::move(working_clause));
    working_clause = Clause();
}

const Variable *RepositoryBuildingVisitor::visit(const MutableVariable &node) const
{
    return symbol_repository->add_symbol<Variable>(
            std::make_unique<Variable>(std::string(node.get_base_name()),
                    std::string(node.get_base_name()) + MutableVariable::disambiguating_delimiter +
                            std::to_string(symbol_repository->get_sentence_count())));
}

const Function *RepositoryBuildingVisitor::visit(MutableFunction &node)
{
    const auto &owned_terms = node.observe_arguments();
    std::vector<const IProcessedTerm *> processed_terms;
    processed_terms.reserve(owned_terms.size());

    for (const auto &term: owned_terms)
        processed_terms.push_back(term->accept(*this));

    return symbol_repository->add_symbol<Function>(std::make_unique<Function>(
            std::string(node.get_disambiguated_name()), std::move(processed_terms)));
}
```

```
const SkolemFunction *RepositoryBuildingVisitor::visit(const MutableSkolemFunction &node)
{
    const auto &quantified_terms = node.observe_arguments();
    std::vector<const IProcessedTerm *> processed_terms;
    processed_terms.reserve(quantified_terms.size());

    for (const auto &term: quantified_terms)
        processed_terms.push_back(term->accept(*this));

    return symbol_repository->add_symbol<SkolemFunction>(std::make_unique<SkolemFunction>(
            std::string(node.get_disambiguated_name()), std::move(processed_terms)));
}

} // namespace optifol
```

1.16.1.2 RepositoryBuildingVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Symbol Repository-Building Visitor
 * @author Oliver Dixon
 * @date 2025-06-22
 * @version Development
 */

#ifndef REPOSITORYBUILDINGVISITOR_HPP
#define REPOSITORYBUILDINGVISITOR_HPP

#include <string_view>

#include "../../IR/Sentences/BinaryConnected.hpp"
#include "../../IR/Sentences/SentenceRoot.hpp"
#include "../../IR/SymbolRepository.hpp"

namespace optifol
{

class MutablePredicate;
class Literal;
class MutableQuantified;
class MutableSentenceRoot;
class SentenceRoot;
class MutableBinaryConnected;
class BinaryConnected;
class MutableVariable;
class Variable;
class MutableFunction;
class Function;
class MutableSkolemFunction;
class SkolemFunction;

class IProcessedSentence;

/**
 * @class RepositoryBuildingVisitor
 * @brief Recursively visits a mutable, owning IMutableSentence/IMutableTerm AST structure and extracts common
 * terms into a baseline SymbolRepository; transforms into an immutable, non-owning ISentence/IProcessedTerm
 * AST.
 * @details
 *  <p>
 *      Logical analysis requires identification of common symbols (predicates, functions, free variables
 * etc.) across all formulas within a problem set. Lexing, parsing, and CNF normalisation does not handle
 * extraction of symbols, and the handling of duplicates thereof, into a common domain repository. The
 * RepositoryBuildingVisitor performs this task as the last stage of the FOL-preparation pipeline. HCI is also
 * substantially improved as common elements can be identified and manipulated with visual aids.
 *  </p>
 *  <p>
 *      The translation of a mutable, owning AST tree into an immutable, non-owning dual achieves large gains
 * in expressiveness and simplicity. Ownership semantics are reduced to a single owning repository (quickly
 * searchable due to hashing properties of immutable nodes and STL @ref std::unordered_set with C++23
 * transparent hashing) that imposes lifetime guarantees on its members. Immutable AST nodes then hold an
 * observing raw pointer to the corresponding item detained by the SymbolRepository, accessible through const
 * noexcept getters, which are aliased in cases of common symbols across nodes and formulas.
 *  </p>
```

```
 *  <p>
 *      In particular, the visitor performs the following transformations on the mutable sentence trees to
 * produce corresponding immutable trees: <table> <tr> <th>Mutable Sentence Target</th> <th>Immutable
 * Production</th> <th>Extractable Terms</th> <th>Extractable Sentences</th>
 *          </tr>
 *          <tr>
 *              <td>MutableBinaryConnected</td>
 *              <td>BinaryConnected</td>
 *              <td>None</td>
 *              <td>LHS sentence; RHS sentence</td>
 *          </tr>
 *          <tr>
 *              <td>MutablePredicate</td>
 *              <td>Predicate</td>
 *              <td>All argument terms</td>
 *              <td>None</td>
 *          </tr>
 *          <tr>
 *              <td>MutableSentenceRoot</td>
 *              <td>SentenceRoot</td>
 *              <td>None</td>
 *              <td>Detained sentence</td>
 *          </tr>
 *      </table>
 *      Note that the MutableSentenceRoot transformation represents a special case: the generated SentenceRoot
 * is not registered in a SymbolRepository but instead passed within an ownership-controlled container, which
 * becomes the responsibility of the visitor. Callers may then extract the last-generated sentence root from
 * the visitor instance, which contains a set of literals ("clauses") under conjunction.
 *  </p>
 *  <p>
 *      Term trees are also eligible for transformation by the visitor:
 *      <table>
 *          <tr>
 *              <th>Mutable Term Target</th>
 *              <th>Immutable Production</th>
 *              <th>Extractable Terms</th>
 *          </tr>
 *          <tr>
 *              <td>MutableFunction</td>
 *              <td>Function</td>
 *              <td>All argument terms</td>
 *          </tr>
 *          <tr>
 *              <td>MutableVariable</td>
 *              <td>Variable</td>
 *              <td>None</td>
 *          </tr>
 *      </table>
 *  </p>
 * @see SymbolRepository
 * @warning The RepositoryBuildingVisitor is a one-way transformation between mutable and immutable
 * representations. Visited nodes are always decomposed and rendered unusable to transfer ownership to the
 * central SymbolRepository.
 */
class RepositoryBuildingVisitor final
{
public:
    /**
     * @brief Create a RepositoryBuildingVisitor with a corresponding SymbolRepository for trans-formula
     * symbol storage
     * @param symbol_repository A mutable SymbolRepository
     */
    explicit RepositoryBuildingVisitor(std::shared_ptr<SymbolRepository> symbol_repository);

    /**
     * @brief Gets the human-readable visitor name
     * @return An observing view of the human-readable visitor name string
     */
    [[nodiscard]] static std::string_view get_visitor_name();

    /**
     * @brief Placeholder visitor to build an immutable quantified item from a MutableQuantified node.
     * @param node The quantified node.
     * @warning
     *  <p>
     *      This method always fails with a bad assertion. Mutable sentences are obliged to accept visitation
     * requests from the RepositoryBuildingVisitor, but at present, the repository only contains expression
     * trees in full Conjunctive Normal Form (CNF). Predicate CNF does not permit quantifiers, and all
     * quantifiers should have been removed by the following visitors in the normalisation pipeline: <ul>
     *        <li>Universal quantifiers: UniversalEliminationVisitor; and</li>
     *        <li>Existential quantifiers: SkolemIntroducingVisitor (where existentials are replaced with
```

```
     * synthesised Skolem functions provided by MutableSkolemFunction).
     *      </ul>
     *  </p>
     * @return @ref std::nullptr_t
     */
    static const IProcessedSentence *visit(const MutableQuantified &node);

    /**
     * @brief Transforms a MutableBinaryConnected disjunctive or conjunctive node into an equivalent
     * BinaryConnected node.
     * @param node The MutableBinaryConnected node to be destroyed and used to construct the BinaryConnected
     * equivalent.
     * @return The BinaryConnected node, referencing symbols in the SymbolRepository equivalent to the
     * operands of the original MutableBinaryConnected.
     * @pre The given MutableBinaryConnected node must be a disjunction or conjunction; the presence of any
     * other operator indicates a non-normalised tree.
     */
    const BinaryConnected *visit(MutableBinaryConnected &node);

    /**
     * @brief Transforms a MutablePredicate node into a Literal, registering the argument terms in the central
     *  SymbolRepository. The Literal itself is also added to the central SymbolRepository.
     * @param node The MutablePredicate node to be transformed into the corresponding immutable
     * representation.
     * @return A non-owning immutable pointer to the SymbolRepository Literal node.
     */
    const Literal *visit(MutablePredicate &node);

    const Variable *visit(const MutableVariable &node) const;

    const Function *visit(MutableFunction &node);

    const SkolemFunction *visit(const MutableSkolemFunction &node);

    void visit(MutableSentenceRoot &node);

    std::unique_ptr<SentenceRoot> take_last_root() noexcept;

private:
    void commit_working_clause();

    /**
     * @brief The working clause stores the working set of literals under disjunction for the current clause.
     * The working clause should be committed to the SentenceRoot @ref root node once it has been fully
     * populated.
     * @see SentenceRoot::commit_clause
     */
    Clause working_clause;

    std::unique_ptr<SentenceRoot> root;

    static const char *visitor_name;

    const std::shared_ptr<SymbolRepository> symbol_repository;
};

} // namespace optifol

#endif
```

1.16.1.3 Observers

1.16.1.3.1 IObservingNodeVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Interface specification for the node-observing visitor.
 * @author Oliver Dixon
 * @date 2024-11-29
 * @version Development
 */

#ifndef IOBSERVINGNODEVISITOR_HPP
#define IOBSERVINGNODEVISITOR_HPP
```

```
namespace optifol
{

class MutableQuantified;
class MutableBinaryConnected;
class MutablePredicate;
class MutableSentenceRoot;

class MutableFunction;
class MutableSkolemFunction;
class MutableVariable;

class IObservingNodeVisitor
{
public:
    virtual ~IObservingNodeVisitor() = default;

    virtual void visit(const MutableQuantified &node) = 0;

    virtual void visit(const MutableBinaryConnected &node) = 0;

    virtual void visit(const MutablePredicate &node) = 0;

    virtual void visit(const MutableSentenceRoot &node) = 0;

    virtual void visit(const MutableFunction &node) = 0;

    virtual void visit(const MutableVariable &node) = 0;
};

} // namespace optifol

#endif
```

1.16.1.3.2 LaTeXSerialisationVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

//
// Created by owd on 7/6/25.
//

#include "LaTeXSerialisationVisitor.hpp"

#include "../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../../../IR/MutableVariants/Sentences/MutablePredicate.hpp"
#include "../../../IR/MutableVariants/Sentences/MutableQuantified.hpp"
#include "../../../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"
#include "../../../IR/MutableVariants/Terms/MutableFunction.hpp"
#include "../../../IR/MutableVariants/Terms/MutableVariable.hpp"

namespace optifol
{

void LaTeXSerialisationVisitor::visit(const MutableSentenceRoot &node)
{
    latex << '$';
    node.observe_sentence()->accept(*this);
    latex << '$';
}

void LaTeXSerialisationVisitor::visit(const MutableBinaryConnected &node)
{
    node.observe_lhs_operand()->accept(*this);
    latex << get_connected_symbol(node.get_operator_type()) << ' ';
    node.observe_rhs_operand()->accept(*this);
}

void LaTeXSerialisationVisitor::visit(const MutableQuantified &node)
{
    latex << get_quantifier_symbol(node.get_quantifier_type()) << ' ';

    node.observe_bound_term()->accept(*this);
    latex << "\\left(";
    node.observe_sentence()->accept(*this);
```

```
    latex << "\\right)";
}

void LaTeXSerialisationVisitor::visit(const MutablePredicate &node)
{
    latex << node.get_name() << "\\left(";
    const auto &arguments = node.observe_arguments();

    if (!arguments.empty()) {
        const auto argument_count = arguments.size() - 1;

        for (std::size_t argument_idx = 0; argument_idx < argument_count; ++argument_idx) {
            arguments[argument_idx]->accept(*this);
            latex << ',';
        }

        arguments[argument_count]->accept(*this);
    }

    latex << "\\right)";
}

void LaTeXSerialisationVisitor::visit(const MutableFunction &node)
{
    latex << node.get_disambiguated_name() << "\\left(";
    const auto &arguments = node.observe_arguments();

    if (!arguments.empty()) {
        const auto argument_count = arguments.size() - 1;

        for (std::size_t argument_idx = 0; argument_idx < argument_count; ++argument_idx) {
            arguments[argument_idx]->accept(*this);
            latex << ',';
        }

        arguments[argument_count]->accept(*this);
    }

    latex << "\\right)";
}

void LaTeXSerialisationVisitor::visit(const MutableVariable &node)
{
    latex << node.to_string();
}

std::string LaTeXSerialisationVisitor::extract() const
{
    return latex.str();
}

const char *LaTeXSerialisationVisitor::get_quantifier_symbol(const QuantifierTypes quantifier) noexcept
{
    switch (quantifier) {
    case QuantifierTypes::Universal:
        return "\\forall";
    case QuantifierTypes::Existential:
        return "\\exists";
    }

    return "??";
}

const char *LaTeXSerialisationVisitor::get_connected_symbol(const BinaryOperatorTypes op) noexcept
{
    switch (op) {
    case BinaryOperatorTypes::Implication:
        return "\\implies";
    case BinaryOperatorTypes::Conjunction:
        return "\\land";
    case BinaryOperatorTypes::Disjunction:
        return "\\lor";
    case BinaryOperatorTypes::Biconditional:
        return "\\iff";
    }

    return "??";
}

} // namespace optifol
```

1.16.1.3.3 LaTeXSerialisationVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

//
// Created by owd on 7/6/25.
//

#ifndef LATEXSERIALISATIONVISITOR_HPP
#define LATEXSERIALISATIONVISITOR_HPP

#include <sstream>

#include "IObservingNodeVisitor.hpp"

namespace optifol
{

enum class QuantifierTypes;
enum class BinaryOperatorTypes;

class LaTeXSerialisationVisitor : public IObservingNodeVisitor
{
public:
    void visit(const MutableSentenceRoot &node) override;

    void visit(const MutableBinaryConnected &node) override;

    void visit(const MutableQuantified &node) override;

    void visit(const MutablePredicate &node) override;

    void visit(const MutableFunction &node) override;

    void visit(const MutableVariable &node) override;

    std::string extract() const;

private:
    [[nodiscard]] static const char *get_quantifier_symbol(QuantifierTypes quantifier) noexcept;

    [[nodiscard]] static const char *get_connected_symbol(BinaryOperatorTypes op) noexcept;

    std::ostringstream latex;
};

} // namespace optifol

#endif // LATEXSERIALISATIONVISITOR_HPP
```

1.16.1.3.4 TextSerialiserVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the text-serialising observing visitor.
 * @author Oliver Dixon
 * @date 2024-11-29
 * @version Development
 */

#include "TextSerialiserVisitor.hpp"

#include "../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../../../IR/MutableVariants/Sentences/MutablePredicate.hpp"
#include "../../../IR/MutableVariants/Sentences/MutableQuantified.hpp"
#include "../../../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"
#include "../../../IR/MutableVariants/Terms/MutableFunction.hpp"
#include "../../../IR/MutableVariants/Terms/MutableVariable.hpp"

namespace optifol
{
```

```
void TextSerialiserVisitor::visit(const MutableQuantified &node)
{
    print_polarity(&node);
    output_stream << MutableQuantified::get_operator_symbol(node.get_quantifier_type())
                  << node.observe_bound_term()->to_string() << ' ';

    node.observe_sentence()->accept(*this);
}

void TextSerialiserVisitor::visit(const MutableBinaryConnected &node)
{
    print_polarity(&node);
    output_stream << '(';

    node.observe_lhs_operand()->accept(*this);
    output_stream << BinaryConnected::get_operator_symbol(node.get_operator_type());
    node.observe_rhs_operand()->accept(*this);

    output_stream << ')';
}

void TextSerialiserVisitor::visit(const MutablePredicate &node)
{
    const auto &arguments = node.observe_arguments();
    print_polarity(&node);
    output_stream << node.get_name() << '(';

    const auto argument_count = arguments.size();
    for (std::remove_const_t<decltype(argument_count)> i = 1; i < argument_count; ++i)
        output_stream << arguments[i - 1]->to_string() << ',' << ' ';

    if (argument_count > 0)
        output_stream << arguments[argument_count - 1]->to_string();

    output_stream << ')';
}

void TextSerialiserVisitor::visit(const MutableSentenceRoot &node)
{
    node.observe_sentence()->accept(*this);
}

void TextSerialiserVisitor::visit(const MutableFunction &node)
{
    const auto &arguments = node.observe_arguments();
    output_stream << node.get_disambiguated_name() << '(';

    const auto argument_count = arguments.size();
    for (std::remove_const_t<decltype(argument_count)> i = 1; i < argument_count; ++i)
        output_stream << arguments[i - 1]->to_string() << ',' << ' ';

    if (argument_count > 0)
        output_stream << arguments[argument_count - 1]->to_string();

    output_stream << ')';
}

void TextSerialiserVisitor::visit(const MutableVariable &node)
{
    output_stream << node.to_string();
}

std::string TextSerialiserVisitor::extract()
{
    auto str = output_stream.str();
    std::ostringstream().swap(output_stream);
    return str;
}

void TextSerialiserVisitor::print_polarity(const IMutableSentence *node)
{
    if (node->is_negative_polarity())
        output_stream << '~';
}

} // namespace optifol
```

1.16.1.3.5 TextSerialiserVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the text-serialising observing visitor.
 * @author Oliver Dixon
 * @date 2024-11-29
 * @version Development
 */

#ifndef TEXTSERIALISERVISITOR_HPP
#define TEXTSERIALISERVISITOR_HPP

#include <sstream>

#include "../../../IR/MutableVariants/Sentences/IMutableSentence.hpp"
#include "IObservingNodeVisitor.hpp"

namespace optifol
{

enum class BinaryOperatorTypes;
enum class QuantifierTypes;

class TextSerialiserVisitor : public IObservingNodeVisitor
{
public:
    void visit(const MutableQuantified &node) override;

    void visit(const MutableBinaryConnected &node) override;

    void visit(const MutablePredicate &node) override;

    void visit(const MutableSentenceRoot &node) override;

    void visit(const MutableFunction &node) override;

    void visit(const MutableVariable &node) override;

    [[nodiscard]] std::string extract();

private:
    std::ostringstream output_stream;

    void print_polarity(const IMutableSentence *node);
};

} // namespace optifol


#endif
```

1.16.1.4 Sentences

1.16.1.4.1 MutatingSentenceVisitorBase.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Mutating Sentence Visitor base class.
 * @author Oliver Dixon
 * @date 2024-11-24
 * @version Development
 */

#include "MutatingSentenceVisitorBase.hpp"

#include "../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../../../IR/MutableVariants/Sentences/MutableQuantified.hpp"
```

```cpp
#include "../../../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"

namespace optifol
{

void MutatingSentenceVisitorBase::visit(MutableQuantified &node)
{
    auto borrowed_sentence = node.take_sentence();
    borrowed_sentence->accept(*this);
    node.put_sentence(std::move(borrowed_sentence));
}

void MutatingSentenceVisitorBase::visit(MutableBinaryConnected &node)
{
    auto borrowed_operand = node.take_lhs_operand();
    borrowed_operand->accept(*this);
    node.put_lhs_operand(std::move(borrowed_operand));

    borrowed_operand = node.take_rhs_operand();
    borrowed_operand->accept(*this);
    node.put_rhs_operand(std::move(borrowed_operand));
}

void MutatingSentenceVisitorBase::visit(MutablePredicate &node)
{
    (void) node;
}

void MutatingSentenceVisitorBase::visit(MutableSentenceRoot &node)
{
    auto borrowed_sentence = node.take_sentence();
    borrowed_sentence->accept(*this);
    node.put_sentence(std::move(borrowed_sentence));
}

} // namespace optifol
```

1.16.1.4.2 MutatingSentenceVisitorBase.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Mutating Sentence Visitor base class.
 * @author Oliver Dixon
 * @date 2024-12-01
 * @version Development
 */

#ifndef MUTATINGSENTENCEVISITORBASE_HPP
#define MUTATINGSENTENCEVISITORBASE_HPP

#include <string_view>

namespace optifol
{

class MutableQuantified;
class MutableBinaryConnected;
class MutablePredicate;
class MutableSentenceRoot;

/**
 * @class MutatingSentenceVisitorBase
 * @brief The Mutating Sentence Visitor Base provides an abstract base for sentence-mutating visitors.
 */
class MutatingSentenceVisitorBase
{
public:
    virtual ~MutatingSentenceVisitorBase() = default;

    [[nodiscard]] virtual std::string_view get_visitor_name() const = 0;

    /**
     * @brief Applies a mutating transformation to the given quantified sentence node, including the
     * quantified sentence.
```

```
     * @param node The quantified sentence root node of the transformation target.
     */
    virtual void visit(MutableQuantified &node);

    /**
     * @brief Applies a mutating transformation to the given connected sentence node and both operands.
     * @param node The connected sentence root node of the transformation target.
     */
    virtual void visit(MutableBinaryConnected &node);

    /**
     * @brief Applies a mutating transformation to the given predicate sentence node and all arguments.
     * @param node The predicate sentence root node of the transformation target.
     */
    virtual void visit(MutablePredicate &node);

    /**
     * @brief Applies a mutating transformation to the given sentence root
     * @param node The sentence root container containing the root of the sentence.
     */
    virtual void visit(MutableSentenceRoot &node);
};

} // namespace optifol

#endif
```

1.16.1.4.3 CNFNormalisers

1.16.1.4.3.1 DisjunctionDistributionVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Disjunction-Distribution Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-25
 * @version Development
 */

#include <cassert>

#include "../../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "DisjunctionDistributionVisitor.hpp"

namespace optifol
{

const char *DisjunctionDistributionVisitor::visitor_name = "DisjunctionDistribution";

std::string_view DisjunctionDistributionVisitor::get_visitor_name() const
{
    return visitor_name;
}

void DisjunctionDistributionVisitor::visit(MutableBinaryConnected &node)
{
    const auto current_operator_type = node.get_operator_type();

    if (current_operator_type == BinaryOperatorTypes::Conjunction)
        /*
         * If we're a conjunction node, we might be a candidate child. Transfer ownership of our LHS and RHS
         * operands to the top layer of the operand stack, with the order determined by the tracking mode, to
         * be returned by the attempt_reduction member function.
         */
        switch (tracking_mode) {
        case TrackingMode::LeftMajor:
            tracked_operands.emplace(node.take_lhs_operand(), node.take_rhs_operand());
            break;

        case TrackingMode::RightMajor:
            tracked_operands.emplace(node.take_rhs_operand(), node.take_lhs_operand());
            break;

        case TrackingMode::NotTracking:
```

```
            // If there's nothing for us to do, pass both operands through the visitor without further action.
            MutatingSentenceVisitorBase::visit(node);
            break;
        }
    else
        // If we're not a child candidate, we mark the end of any chain. Stop the tracking here.
        tracking_mode = TrackingMode::NotTracking;

    if (current_operator_type == BinaryOperatorTypes::Disjunction) {
        /*
         * If we're a disjunction node, we might be a candidate parent. Check the left and right branches for
         * candidate children, reducing recursively where necessary. Once we've reduced on both branches, and
         * ownership has been returned, stop the tracking.
         */

        tracking_mode = TrackingMode::RightMajor;
        auto borrowed_rhs = node.take_rhs_operand();
        borrowed_rhs->accept(*this);
        node.put_rhs_operand(std::move(borrowed_rhs));

        if (!attempt_reduction(node)) {
            // If we can't do a reduction on the RHS, try the LHS.

            tracking_mode = TrackingMode::LeftMajor;
            auto borrowed_lhs = node.take_lhs_operand();
            borrowed_lhs->accept(*this);
            node.put_lhs_operand(std::move(borrowed_lhs));

            attempt_reduction(node);
        }

        tracking_mode = TrackingMode::NotTracking;
    }
}

bool DisjunctionDistributionVisitor::attempt_reduction(MutableBinaryConnected &node)
{
    // Candidate children are, by definition, disjunctive clauses.
    assert(node.get_operator_type() == BinaryOperatorTypes::Disjunction);

    if (!tracked_operands.empty()) {
        /*
         * If the tracked operands stack is non-empty, it still holds ownership of operands in clauses that
         * need to be distributed. We consider four 'destination' operands based on the top layer of the stack
         * used to construct the distributed conjunctive clause of disjuncts:
         *
         *  - LHS/LHS: The LHS operand of the first disjunct
         *  - LHS/RHS: The RHS operand of the first disjunct
         *  - RHS/LHS: The LHS operand of the second disjunct. This is the distributed literal cloned from
         * LHS/LHS.
         *  - RHS/RHS: The RHS operand of the second disjunct.
         */

        auto destination_lhs_lhs =
                tracking_mode == TrackingMode::LeftMajor ? node.take_rhs_operand() : node.take_lhs_operand();
        auto destination_lhs_rhs = std::move(tracked_operands.top().first);

        auto destination_rhs_lhs = destination_lhs_lhs->clone();
        auto destination_rhs_rhs = std::move(tracked_operands.top().second);

        node.set_operator_type(BinaryOperatorTypes::Conjunction);

        node.put_lhs_operand(std::make_unique<MutableBinaryConnected>(BinaryOperatorTypes::Disjunction,
                std::move(destination_lhs_lhs), std::move(destination_lhs_rhs)));

        node.put_rhs_operand(std::make_unique<MutableBinaryConnected>(BinaryOperatorTypes::Disjunction,
                std::move(destination_rhs_lhs), std::move(destination_rhs_rhs)));

        tracked_operands.pop();
        MutatingSentenceVisitorBase::visit(node);

        /*
         * The above recursive call should empty the tracked operands stack with this member function. If we
         * end with a non-empty stack, it still owns operands that should've been returned to the
         * MutableBinaryConnected or used to construct a new operand.
         */
        assert(tracked_operands.empty());
        return true;
    }

    return false;
```

```
}

} // namespace optifol
```

1.16.1.4.3.2 DisjunctionDistributionVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Disjunction-Distribution Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-25
 * @version Development
 */

#ifndef DISJUNCTIONDISTRIBUTIONVISITOR_HPP
#define DISJUNCTIONDISTRIBUTIONVISITOR_HPP

#include <memory>
#include <stack>

#include "../MutatingSentenceVisitorBase.hpp"

namespace optifol
{

class IMutableSentence;

/**
 * @class DisjunctionDistributionVisitor
 * @brief The Disjunction-Distribution Visitor applies the distributive law to eligible connected sentences.
 *
 * @details The Disjunction-Distribution Visitor rewrites the model to distribute disjunctions across clauses
 * of nested conjunctions. In particular, <ul> <li><code>P | (Q & R)</code> becomes <code>(P | Q) & (P |
 * R)</code>; and likewise, </li> <li><code>(P & Q) | R</code> becomes <code>(R | P) & (R | Q)</code>.</li>
 * </ul>
 * The rewriting rules executed herein do not make use of proxies, as fundamental types are not altered (i.e.
 * only the substance of the MutableBinaryConnected operands are altered).
 *
 * @warning Although multiple passes are not required for this CNF-normalising visitor, it does recurse on any
 * produced terms to ensure a full reduction. On extremely deeply nested sentences, this could cause a machine
 * stack overflow.
 */
class DisjunctionDistributionVisitor : public MutatingSentenceVisitorBase
{
public:
    [[nodiscard]] std::string_view get_visitor_name() const override;

    /**
     * @brief Recursively applies disjunction-distribution to the connected sentence, in-place.
     * @param node The root of the connected sentence on which disjunction-distribution should be applied.
     */
    void visit(MutableBinaryConnected &node) override;

private:
    /**
     * @enum TrackingMode
     * @brief Indicate the current state of 'tracking', as required by a calling visitor.
     * @details When tracking is enabled (left- or right-major), operands/children of binary-connected nodes
     * should be tracked by the DisjunctionDistributionVisitor instance.
     */
    enum class TrackingMode
    {
        NotTracking, /**< Not tracking; nested children shouldn't record their operands. */
        LeftMajor, /**< Tracking to the left: nested children should record their left operands in the major
                      slot */
        RightMajor /**< Tracking to the right: nested children should record their right operands in the major
                      slot */
    };

    static const char *visitor_name;

    /**
     * @brief The current operand-tracking state
     */
```

```
    TrackingMode tracking_mode = TrackingMode::NotTracking;

    /**
     * @brief The tracked operand stack stores, depth-wise, the major and minor child operands of connected
     * nodes, respectively.
     */
    std::stack<std::pair<std::unique_ptr<IMutableSentence>, std::unique_ptr<IMutableSentence>>>
            tracked_operands;

    /**
     * @brief Apply any applicable reductions to the given node, given the collected tracked operands from
     * eligible children.
     * @pre The given node must be of a disjunctive nature.
     * @post The tracked operand stack is empty.
     * @param node The root node on which reduction should be applied
     * @return Was at least one reduction performed?
     */
    bool attempt_reduction(MutableBinaryConnected &node);
};

} // namespace optifol

#endif
```

1.16.1.4.3.3 DMLVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the De Morgan's Laws Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-27
 * @version Development
 */

#include "DMLVisitor.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableQuantified.hpp"

namespace optifol
{

const char *DMLVisitor::visitor_name = "DeMorgan";

std::string_view DMLVisitor::get_visitor_name() const
{
    return visitor_name;
}

void DMLVisitor::visit(MutableBinaryConnected &node)
{
    /*
     * If we're in an immediately negative context ~(MutableBinaryConnected), we apply the transformation
     * according to the binary operator type. This requires taking temporary ownership of the operands.
     */
    if (node.is_negative_polarity()) {
        auto borrowed_lhs = node.take_lhs_operand();
        auto borrowed_rhs = node.take_rhs_operand();

        switch (node.get_operator_type()) {
        case BinaryOperatorTypes::Disjunction:
            node.set_operator_type(BinaryOperatorTypes::Conjunction);
            borrowed_lhs->flip_polarity();
            borrowed_rhs->flip_polarity();
            node.flip_polarity();
            break;

        case BinaryOperatorTypes::Conjunction:
            node.set_operator_type(BinaryOperatorTypes::Disjunction);
            borrowed_lhs->flip_polarity();
            borrowed_rhs->flip_polarity();
            node.flip_polarity();
            break;

        default:
```

```
            (void) 0;
        }

        node.put_lhs_operand(std::move(borrowed_lhs));
        node.put_rhs_operand(std::move(borrowed_rhs));
    }

    /*
     * Regardless of whether we did a transformation, there may be further opportunities for transformation on
     * the individual operands.
     */
    MutatingSentenceVisitorBase::visit(node);
}

void DMLVisitor::visit(MutableQuantified &node)
{
    /*
     * If we're in an immediately negative context ~(MutableQuantified), we apply the transformation according
     * to the quantification type. This requires taking temporary ownership of the quantified sentence.
     */
    if (node.is_negative_polarity()) {
        auto borrowed_sentence = node.take_sentence();

        switch (node.get_quantifier_type()) {
        case QuantifierTypes::Universal:
            node.set_quantifier_type(QuantifierTypes::Existential);
            borrowed_sentence->flip_polarity();
            break;

        case QuantifierTypes::Existential:
            node.set_quantifier_type(QuantifierTypes::Universal);
            borrowed_sentence->flip_polarity();
            break;
        }

        node.put_sentence(std::move(borrowed_sentence));
        node.flip_polarity();
    }

    /*
     * Regardless of whether we did a transformation, there may be further opportunities for transformation on
     * the quantified sentence.
     */
    MutatingSentenceVisitorBase::visit(node);
}

} // namespace optifol
```

1.16.1.4.3.4 DMLVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the De Morgan's Laws Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-27
 * @version Development
 */

#ifndef DMLVISITOR_HPP
#define DMLVISITOR_HPP

#include <memory>
#include <vector>

#include "../MutatingSentenceVisitorBase.hpp"

namespace optifol
{

class IMutableSentence;

/**
 * @class DMLVisitor
 * @brief The DML (De Morgan's Laws) Visitor applies De Morgan's Laws with negative reduction to eligible
```

```
 * sentences.
 *
 * @details The DML visitor recursively rewrites the root model with the following rules:
 * <ul>
 *   <li><code>~(P | Q)</code> becomes <code>~P & ~Q</code>;</li>
 *   <li><code>~(P & Q)</code> becomes <code>~P | ~Q</code>;</li>
 *   <li><code>~~P</code> becomes <code>P</code>;</li>
 *   <li><code>~%Ux(P(x))</code> becomes <code>%Ex(~P(x))</code>; and</li>
 *   <li><code>~%Ex(P(x))</code> becomes <code>%Ux(~P(x))</code>.</li>
 * </ul>
 *
 * @warning Although multiple passes are not required for this DML-normalising visitor, it does recurse on any
 * produced terms to ensure a full reduction. On extremely deeply nested sentences, this could cause a machine
 * stack overflow.
 */
class DMLVisitor : public MutatingSentenceVisitorBase
{
public:
    [[nodiscard]] std::string_view get_visitor_name() const override;

    void visit(MutableBinaryConnected &node) override;

    void visit(MutableQuantified &node) override;

private:
    static const char *visitor_name;
};

} // namespace optifol

#endif
```

1.16.1.4.3.5 ImplicationEliminationVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Implication-Elimination Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-24
 * @version Development
 */

#include "ImplicationEliminationVisitor.hpp"

#include "../../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"

namespace optifol
{

const char *ImplicationEliminationVisitor::visitor_name = "ImplicationElimination";

std::string_view ImplicationEliminationVisitor::get_visitor_name() const
{
    return visitor_name;
}

void ImplicationEliminationVisitor::visit(MutableBinaryConnected &node)
{
    MutatingSentenceVisitorBase::visit(node);

    const auto operator_type = node.get_operator_type();

    if (operator_type == BinaryOperatorTypes::Implication) {
        node.set_operator_type(BinaryOperatorTypes::Disjunction);
        auto borrowed_lhs = node.take_lhs_operand();
        borrowed_lhs->flip_polarity();
        node.put_lhs_operand(std::move(borrowed_lhs));
    } else if (operator_type == BinaryOperatorTypes::Biconditional) {
        node.set_operator_type(BinaryOperatorTypes::Conjunction);

        auto save_lhs = node.take_lhs_operand();
        auto save_rhs = node.take_rhs_operand();

        auto new_rhs = save_rhs->clone();
```

```
        new_rhs->flip_polarity();

        node.put_lhs_operand(std::make_unique<MutableBinaryConnected>(
                BinaryOperatorTypes::Disjunction, save_lhs->clone(), std::move(new_rhs)));

        save_lhs->flip_polarity();

        node.put_rhs_operand(std::make_unique<MutableBinaryConnected>(
                BinaryOperatorTypes::Disjunction, std::move(save_lhs), std::move(save_rhs)));
    }
}

} // namespace optifol
```

1.16.1.4.3.6 ImplicationEliminationVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Implication-Elimination Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-24
 * @version Development
 */

#ifndef IMPLICATIONELIMINATIONVISITOR_HPP
#define IMPLICATIONELIMINATIONVISITOR_HPP

#include "../MutatingSentenceVisitorBase.hpp"

namespace optifol
{

/**
 * @class ImplicationEliminationVisitor
 * @brief The Implication Elimination Visitor applies the first stage of in-situ model-rewriting to Negated
 * Normal Form.
 *
 * @details The Implication Elimination Visitor rewrites the model to remove implications and equivalences.
 * Implications are rewritten as disjunctions, and equivalences are rewritten as conjunctions of disjunctions.
 * In particular, <ul> <li><code>P(x) =&gt; Q(x)</code> becomes <code>~P(x) | Q(x)</code>; and likewise, </li>
 *  <li><code>P(x) &lt;=&gt; Q(x)</code> becomes <code>(P(x) | ~Q(x)) & (~P(x) | Q(x))</code>.</li>
 * </ul>
 * The rewriting rules executed herein do not make use of proxies, as fundamental types are not altered (i.e.
 * only the substance of the MutableBinaryConnected operands are altered).
 */
class ImplicationEliminationVisitor : public MutatingSentenceVisitorBase
{
public:
    [[nodiscard]] std::string_view get_visitor_name() const override;

    /**
     * @brief Recursively applies implication-elimination transformations to the given connected sentence root
     * @param node The connected sentence root node on which the implication-elimination should be applied
     */
    void visit(MutableBinaryConnected &node) override;

private:
    static const char *visitor_name;
};

} // namespace optifol

#endif
```

1.16.1.4.3.7 QuantifierExtractingVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
```

```cpp
 * @file
 * @brief Class implementation for the Quantifier Extracting Visitor
 * @author Oliver Dixon
 * @date 2025-05-04
 * @version Development
 */

#include <cassert>

#include "QuantifierExtractingVisitor.hpp"

#include "../../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"

namespace optifol
{

const char *QuantifierExtractingVisitor::visitor_name = "QuantifierExtraction";

std::string_view QuantifierExtractingVisitor::get_visitor_name() const
{
    return visitor_name;
}

void QuantifierExtractingVisitor::visit(MutableBinaryConnected &node)
{
    assert(!transformation.has_value()); // Ensure there is no pending transformation.

    // Recurse down the LHS and apply any applicable transforms generated during the visit.
    tracking_mode = TrackingMode::LeftMajor;
    auto borrowed_operand_lhs = node.take_lhs_operand();
    borrowed_operand_lhs->accept(*this);
    borrowed_operand_lhs = apply_transform(std::move(borrowed_operand_lhs));

    if (!quant_lhs_data.has_value())
        // If no quantifier, just return the borrowed LHS.
        node.put_lhs_operand(std::move(borrowed_operand_lhs));

    // Recurse down the RHS and apply any applicable transforms generated during the visit.
    tracking_mode = TrackingMode::RightMajor;
    auto borrowed_operand_rhs = node.take_rhs_operand();
    borrowed_operand_rhs->accept(*this);
    borrowed_operand_rhs = apply_transform(std::move(borrowed_operand_rhs));

    if (!quant_rhs_data.has_value())
        // If no quantifier, just return the borrowed RHS.
        node.put_rhs_operand(std::move(borrowed_operand_rhs));

    assert(!transformation.has_value()); // Ensure there is still no pending transformation.
    tracking_mode = TrackingMode::NotTracking;

    /*
     * Apply the rules:
     *
     *  - If there's a quantifier on the LHS and the RHS, the quantifiers cannot be moved any further
     * outwards.
     *  - If there's neither of the operands were quantifiers, there are no quantifiers to move outwards.
     *  - If there's a quantifier on either the LHS or the RHS, prepare the current MutableBinaryConnected to
     * be the quantified sentence by retaining the non-quantified side, and using the previously quantified
     * sentence as the other operand.
     *
     *  At each stage, we ensure that any borrowed sentences/operands have been returned to an owner. Once
     * this operation is complete, only the bound variable may be held in the quantifier template without a
     * permanent owner.
     */

    if (quant_lhs_data.has_value()) {
        if (!quant_rhs_data.has_value()) {
            // If the LHS only is a quantifier, set our LHS as its sentence.
            node.put_lhs_operand(std::move(quant_lhs_data->sentence));
            transformation.emplace(quant_lhs_data->type, std::move(quant_lhs_data->bound_term));
        } else {
            // If both operands were quantifiers, there's no transformation to do. Return operands to our
            // node.
            node.put_lhs_operand(std::move(borrowed_operand_lhs));
            node.put_rhs_operand(std::move(borrowed_operand_rhs));

            quant_lhs_data->return_to_owner();
            quant_rhs_data->return_to_owner();

            quant_rhs_data.reset();
```

```
        }

        quant_lhs_data.reset();
    } else if (quant_rhs_data.has_value()) {
        // If the RHS only is a quantifier, set our RHS as its sentence.
        node.put_rhs_operand(std::move(quant_rhs_data->sentence));
        transformation.emplace(quant_rhs_data->type, std::move(quant_rhs_data->bound_term));
        quant_rhs_data.reset();
    }

    // Ensure there are no values with indeterminate owners.
    assert(node.observe_lhs_operand() != nullptr);
    assert(node.observe_rhs_operand() != nullptr);
    assert(!quant_lhs_data.has_value());
    assert(!quant_rhs_data.has_value());
}

void QuantifierExtractingVisitor::visit(MutableQuantified &node)
{
    const auto was_tracking = tracking_mode;
    tracking_mode = TrackingMode::NotTracking;

    auto borrowed_sentence = node.take_sentence();
    borrowed_sentence->accept(*this);
    borrowed_sentence = apply_transform(std::move(borrowed_sentence));

    switch (was_tracking) {
    case TrackingMode::NotTracking:
        node.put_sentence(std::move(borrowed_sentence));
        return;

    case TrackingMode::LeftMajor:
        quant_lhs_data.emplace(
                node.get_quantifier_type(), node.take_bound_term(), std::move(borrowed_sentence), &node);
        break;

    case TrackingMode::RightMajor:
        quant_rhs_data.emplace(
                node.get_quantifier_type(), node.take_bound_term(), std::move(borrowed_sentence), &node);
        break;
    }
}

void QuantifierExtractingVisitor::visit(MutableSentenceRoot &node)
{
    auto borrowed_sentence = node.take_sentence();
    borrowed_sentence->accept(*this);
    node.put_sentence(apply_transform(std::move(borrowed_sentence)));
}

std::unique_ptr<IMutableSentence> QuantifierExtractingVisitor::apply_transform(
        std::unique_ptr<IMutableSentence> &&transform_target)
{
    if (transformation.has_value()) {
        transform_target = std::make_unique<MutableQuantified>(
                transformation->first, std::move(transformation->second), std::move(transform_target));
        transformation.reset();

        // The introduction of a new quantified sentence may open new opportunities for reduction, to an
        // arbitrary depth
        transform_target->accept(*this);
    }

    return transform_target;
}

} // namespace optifol
```

1.16.1.4.3.8 QuantifierExtractingVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Quantifier Extracting Visitor
 * @author Oliver Dixon
```

```
 * @date 2025-05-04
 * @version Development
 */

#ifndef QUANTIFIEREXTRACTINGVISITOR_HPP
#define QUANTIFIEREXTRACTINGVISITOR_HPP

#include <memory>
#include <optional>

#include "../../../../IR/MutableVariants/Sentences/IMutableSentence.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableQuantified.hpp"
#include "../../../../IR/MutableVariants/Terms/IMutableTerm.hpp"
#include "../MutatingSentenceVisitorBase.hpp"

namespace optifol
{

/**
 * @class QuantifierExtractingVisitor
 * @brief The quantifier-extracting visitor shifts quantifiers outwards of binary-connected nodes.
 *
 * @details
 *  <p>
 *      The quantifier-extracting visitor rewrites the model such that quantifiers may only occur in the
 * initial prefix of a formula, but never inside a negation, conjunction, or disjunction. In particular, the
 * following transformations are repeatedly applied: <table> <tr> <th>#</th> <th>Match</th>
 *              <th>Replacement</th>
 *          </tr>
 *          <tr>
 *              <td>1</td>
 *              <td>@f$ P \land \forall x Q(x) @f$</td>
 *              <td>@f$ \forall x \left( P \land Q(x) \right) @f$</td>
 *          </tr>
 *          <tr>
 *              <td>2</td>
 *              <td>@f$ P \lor \forall x Q(x) @f$</td>
 *              <td>@f$ \forall x \left( P \lor Q(x) \right) @f$</td>
 *          </tr>
 *          <tr>
 *              <td>3</td>
 *              <td>@f$ P \land \exists x Q(x) @f$</td>
 *              <td>@f$ \exists x \left( P \land Q(x) \right) @f$</td>
 *          </tr>
 *          <tr>
 *              <td>4</td>
 *              <td>@f$ P \lor \exists x Q(x) @f$</td>
 *              <td>@f$ \exists x \left( P \lor Q(x) \right) @f$</td>
 *          </tr>
 *      </table>
 *  </p>
 *  <p>
 *      Note that the above transformations are commutative in the binary operator. That is, the above
 * transformations match equally for quantifiers on the LHS and non-quantifiers on the RHS. Cruically, binary
 * sentences with quantifiers in both or neither operand slot(s) are not eligible for transformation.
 *  </p>
 */
class QuantifierExtractingVisitor : public MutatingSentenceVisitorBase
{
public:
    [[nodiscard]] std::string_view get_visitor_name() const override;

    void visit(MutableBinaryConnected &node) override;

    void visit(MutableQuantified &node) override;

    void visit(MutableSentenceRoot &node) override;

private:
    /**
     * @class QuantifiedTemplate
     * @brief A basic owning structure for quantifier data (type, bound term, and bound sentence) with an
     * owner to which it can be returned.
     */
    struct QuantifiedTemplate
    {
        /**
         * @brief Construct a new QuantifiedTemplate to hold ownership of the given bound term and sentence,
         * borrowed from the specified owner
         * @param type The characteristic type of the quantifier
         * @param bound_term The variable term bound by the quantifier, to be owned by the template
```

```
     * @param sentence The sentence bound by the quantifier, to be owned by the template
     * @param owner The owner of the bound term and sentence to which responsibility may be returned at
     * any point throughout the lifetime of the QuantifiedTemplate.
     */
    QuantifiedTemplate(const QuantifierTypes type, std::unique_ptr<IMutableTerm> &&bound_term,
            std::unique_ptr<IMutableSentence> &&sentence, MutableQuantified *owner) :
        type(type),
        bound_term(std::move(bound_term)),
        sentence(std::move(sentence)),
        owner(owner)
    {
    }

    /**
     * @brief Return the borrowed elements to the responsibility of the original stated owner
     */
    void return_to_owner()
    {
        owner->put_sentence(std::move(sentence));
        owner->put_bound_term(std::move(bound_term));
    }

    QuantifierTypes type;
    std::unique_ptr<IMutableTerm> bound_term;
    std::unique_ptr<IMutableSentence> sentence;

private:
    MutableQuantified *const owner;
};

/**
 * @enum TrackingMode
 * @brief Indicate the current state of 'tracking', as required by a calling visitor.
 * @details When tracking is enabled (left- or right-major), operands/children of binary-connected nodes
 * should be tracked by the QuantifierExtractingVisitor instance.
 */
enum class TrackingMode
{
    NotTracking, /**< Not tracking; nested children shouldn't record their operands. */
    LeftMajor, /**< Tracking to the left: nested children should record their left operands in the major
                  slot */
    RightMajor /**< Tracking to the right: nested children should record their right operands in the major
                  slot */
};

/**
 * @brief Apply any pending quantification transformation to the given (borrowed) target
 * @param transform_target The target sentence to be transformed/wrapped by the pending transformation
 * quantifier
 * @return An owning container to the wrapped target sentence, or the original sentence if there was no
 * suitable pending transform.
 * @details There are two possible cases:
 *  <ol>
 *      <li>
 *          There is no pending transformation. This function does nothing, and returns an owning
 * container to the borrowed target.
 *      </li>
 *      <li>
 *          <p>
 *              There is a pending transformation encoding a type and bound term. The transformation
 * target is interpreted as a sentence that should be wrapped by a MutableQuantifierNode with the
 * character and bound variable described by the transformation pair. The transformation is applied to the
 *              target, thus making it a MutableQuantifierNode binding the sentence previously held by the
 *              target, and the owning container is returned.
 *          </p>
 *          <p>
 *              In general, if the transformation metadata holds a <code>T</code>-type quantifier binding
 * the variable <code>x</code>, and the transformation target holds a sentence <code>S</code>, the
 *              returned value is an owning container for a <code>T</code>-type quantifier binding the
 * variable <code>x</code> with the sentence <code>S</code>.
 *          </p>
 *      </li>
 *  </ol>
 */
std::unique_ptr<IMutableSentence> apply_transform(std::unique_ptr<IMutableSentence> &&transform_target);

static const char *visitor_name;

TrackingMode tracking_mode = TrackingMode::NotTracking;
std::optional<QuantifiedTemplate> quant_lhs_data;
std::optional<QuantifiedTemplate> quant_rhs_data;
```

```
    std::optional<std::pair<QuantifierTypes, std::unique_ptr<IMutableTerm>>> transformation;
};

} // namespace optifol

#endif
```

1.16.1.4.3.9 SkolemIntroducingVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Existential Shifting Visitor
 * @author Oliver Dixon
 * @date 2025-05-04
 * @version Development
 */

#include "SkolemIntroducingVisitor.hpp"

#include <cassert>

#include "../../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutablePredicate.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableQuantified.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"
#include "../../../../IR/MutableVariants/Terms/MutableSkolemFunction.hpp"

namespace optifol
{

const char *SkolemIntroducingVisitor::visitor_name = "SkolemIntroduction";

SkolemIntroducingVisitor::SkolemIntroducingVisitor()
{
    universally_quantified_variables.emplace();
}

SkolemIntroducingVisitor::~SkolemIntroducingVisitor()
{
    assert(skolem_replacements.empty());
    assert(universally_quantified_variables.size() == 1 && universally_quantified_variables.top().empty());
}

std::string_view SkolemIntroducingVisitor::get_visitor_name() const
{
    return visitor_name;
}

void SkolemIntroducingVisitor::visit(MutableQuantified &node)
{
    auto borrowed_sentence = node.take_sentence();

    switch (node.get_quantifier_type()) {
    case QuantifierTypes::Universal:
        open_scope(node);
        borrowed_sentence->accept(*this);
        close_latest_scope(node);

        if (extracted_sentence == nullptr)
            node.put_sentence(std::move(borrowed_sentence));
        else
            node.put_sentence(std::move(extracted_sentence));

        break;

    case QuantifierTypes::Existential:
        eliminate_existential(*node.observe_bound_term());
        borrowed_sentence->accept(*this);
        extracted_sentence = std::move(borrowed_sentence);

        break;
    }
}
```

```
void SkolemIntroducingVisitor::visit(MutablePredicate &node)
{
    if (skolem_replacements.empty())
        return;

    auto &args = node.observe_arguments();
    const auto argument_count = args.size();

    for (std::remove_const_t<decltype(argument_count)> i = 0; i < argument_count; ++i) {
        const auto &potential_replacement =
                skolem_replacements.find(std::string(args[i]->get_disambiguated_name()));
        if (potential_replacement != skolem_replacements.cend())
            args[i] = potential_replacement->second->clone();

        args[i]->accept(term_visitor);
    }
}

void SkolemIntroducingVisitor::visit(MutableBinaryConnected &node)
{
    auto borrowed_operand = node.take_lhs_operand();
    borrowed_operand->accept(*this);
    if (extracted_sentence == nullptr)
        node.put_lhs_operand(std::move(borrowed_operand));
    else
        node.put_lhs_operand(std::move(extracted_sentence));

    borrowed_operand = node.take_rhs_operand();
    borrowed_operand->accept(*this);
    if (extracted_sentence == nullptr)
        node.put_rhs_operand(std::move(borrowed_operand));
    else
        node.put_rhs_operand(std::move(extracted_sentence));
}

void SkolemIntroducingVisitor::visit(MutableSentenceRoot &node)
{
    auto borrowed_sentence = node.take_sentence();
    borrowed_sentence->accept(*this);
    if (extracted_sentence == nullptr)
        node.put_sentence(std::move(borrowed_sentence));
    else
        node.put_sentence(std::move(extracted_sentence));

    skolem_replacements.clear();
}

void SkolemIntroducingVisitor::open_scope(MutableQuantified &node)
{
    universally_quantified_variables.top().push_back(node.take_bound_term());
}

void SkolemIntroducingVisitor::close_latest_scope(MutableQuantified &node)
{
    const auto scoped_var_count = universally_quantified_variables.top().size();
    assert(scoped_var_count >
           0); // We assume to be within a scope, and thus must have at least one bound variable.

    // Return the latest variable from the latest scope to the given node, assumed to be its original owner.
    node.put_bound_term(std::move(universally_quantified_variables.top().back()));

    if (scoped_var_count == 1) {
        // If there's only one variable currently scoped, we clear this scope level and reset it.
        universally_quantified_variables.pop();
        universally_quantified_variables.emplace();
    } else
        /*
         * Otherwise, just remove the returned variable. There are other universally quantified variables that
         * need to be returned at this scope level.
         */
        universally_quantified_variables.top().pop_back();
}

void SkolemIntroducingVisitor::eliminate_existential(const IMutableTerm &target_bound_variable)
{
    // Clone the universally quantified variables in the current scope.
    const auto &arguments = universally_quantified_variables.top();
    std::vector<std::unique_ptr<IMutableTerm>> argument_copy;
    argument_copy.reserve(arguments.size());

    for (const auto &argument: arguments)
```

```
        argument_copy.push_back(argument->clone());

    // Create a new Skolem function to be parameterised by all universally quantified variables in the current
    // scope.
    auto skolem = std::make_unique<MutableSkolemFunction>(
            'S' + std::to_string(skolem_counter++), std::move(argument_copy));

    /*
     * Indicate to the visitor that all instances of the existentially quantified variable should be replaced
     * by a Skolem function, to be cloned from the templated created above.
     */
    skolem_replacements.emplace(target_bound_variable.get_disambiguated_name(), std::move(skolem));
}

} // namespace optifol
```

1.16.1.4.3.10 SkolemIntroducingVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Existential Shifting Visitor
 * @author Oliver Dixon
 * @date 2025-05-04
 * @version Development
 */

#ifndef SKOLEMINTRODUCINGVISITOR_HPP
#define SKOLEMINTRODUCINGVISITOR_HPP

#include <stack>
#include <unordered_map>
#include <vector>

#include "../../../../IR/MutableVariants/Sentences/IMutableSentence.hpp"
#include "../../Terms/TermResolutionVisitor.hpp"
#include "../MutatingSentenceVisitorBase.hpp"

namespace optifol
{

class SkolemIntroducingVisitor : public MutatingSentenceVisitorBase
{
public:
    SkolemIntroducingVisitor();

    ~SkolemIntroducingVisitor() override;

    [[nodiscard]] std::string_view get_visitor_name() const override;

    void visit(MutableQuantified &node) override;

    void visit(MutablePredicate &node) override;

    void visit(MutableBinaryConnected &node) override;

    void visit(MutableSentenceRoot &node) override;

private:
    void open_scope(MutableQuantified &node);

    void close_latest_scope(MutableQuantified &node);

    void eliminate_existential(const IMutableTerm &target_bound_variable);

    static const char *visitor_name;

    std::stack<std::vector<std::unique_ptr<IMutableTerm>>> universally_quantified_variables;

    std::size_t skolem_counter = 0;

    std::unordered_map<std::string, std::unique_ptr<IMutableTerm>, StringHash, std::equal_to<>>
            skolem_replacements;

    TermResolutionVisitor term_visitor{skolem_replacements};
```

```
    std::unique_ptr<IMutableSentence> extracted_sentence;
};

} // namespace optifol

#endif
```

1.16.1.4.3.11 SymbolStandardisingVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Symbol-Standardising Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-12-01
 * @version Development
 */

#include <cassert>

#include "SymbolStandardisingVisitor.hpp"

#include <algorithm>

#include "../../../../Exceptions/SemanticException.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutablePredicate.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableQuantified.hpp"
#include "../../../../IR/MutableVariants/Terms/MutableVariable.hpp"

namespace optifol
{

const char *SymbolStandardisingVisitor::visitor_name = "SymbolStandardiser";

std::string_view SymbolStandardisingVisitor::get_visitor_name() const
{
    return visitor_name;
}

void SymbolStandardisingVisitor::visit(MutableQuantified &node)
{
    /*
     * Open the scope, deal with the contents, and close it. Note that the act of opening a scope transfers
     * ownership of the bound term to the internal visitor state, hence it cannot be used until the scope is
     * closed and the bound term is returned. It is invariant that
     * SymbolStandardisingVisitor::visit(MutableQuantified&) does attempt to access its own bound term while
     * its relevant scope is open.
     */
    const auto rule_reference = open_scope(node);
    MutatingSentenceVisitorBase::visit(node);
    close_scope(node, rule_reference);
}

void SymbolStandardisingVisitor::visit(MutablePredicate &node)
{
    auto &args = node.observe_arguments();
    const auto argument_count = args.size();

    // For each term argument, apply any relevant disambiguation rewriting. Then recurse with the term
    // visitor.
    for (std::remove_const_t<decltype(argument_count)> i = 0; i < argument_count; ++i) {
        const auto &rule = rewriting_rules.find(args[i]->get_disambiguated_name());
        if (rule != rewriting_rules.cend())
            args[i] = rule->second->clone();

        args[i]->accept(term_visitor);
    }
}

std::optional<decltype(SymbolStandardisingVisitor::rewriting_rules)::iterator>
SymbolStandardisingVisitor::open_scope(MutableQuantified &node)
{
    const auto &original_name = node.observe_bound_term()->to_string();
```

```
    if (scope.contains(original_name))
        throw SemanticException(
                "Declared variable \"" + original_name + "\" is already defined in the current scope.");

    if (adjacent.contains(original_name)) {
        // If an adjacent scope has already used a variable with this name, it is ambiguous and needs
        // renaming.
        auto new_name = generate_name(original_name);

        while (adjacent.contains(new_name))
            // Repeatedly disambiguate until we have something unique.
            new_name = generate_name(new_name);

        /* In addition to updating the scope set, we also manage the rewriting rules table, since an entry
         * would only appear given a prospectively ambiguous variable node, which clashes with a bound
         * variable in an adjacent scope. */
        node.put_bound_term(std::make_unique<MutableVariable>(original_name, new_name));
        scope.emplace(node.observe_bound_term()->to_string());
        return rewriting_rules.emplace(original_name, node.take_bound_term()).first;
    }

    scope.emplace(node.observe_bound_term()->to_string());
    return {};
}

void SymbolStandardisingVisitor::close_scope(MutableQuantified &node,
        const std::optional<decltype(SymbolStandardisingVisitor::rewriting_rules)::iterator> &rule_reference)
{
    if (rule_reference.has_value() && rule_reference != rewriting_rules.end()) {
        /*
         * If we were provided with a valid rule iterator reference, return the borrowed bound term to the
         * quantifier. If the optional container is empty, we assume that no rewriting rule was produced for
         * this scope, and thus there was no borrowed bound term that needs to be returned. This is verified
         * with an assert to ensure that we can get some observing reference to the bound term of the given
         * quantified node.
         */
        node.put_bound_term(std::move(rule_reference->operator->()->second));
        rewriting_rules.erase(*rule_reference);
    }

    const auto bound_term = node.observe_bound_term();
    assert(bound_term !=
            nullptr); // Ensure that the bound term has been correctly returned to the quantifier node.

    const auto &name = bound_term->to_string();
    assert(scope.contains(name)); // Ensure that the current scope is what we expect.

    scope.erase(name);
    adjacent.emplace(bound_term->get_disambiguated_name());
}

std::string SymbolStandardisingVisitor::generate_name(const std::string &name)
{
    return name + MutableVariable::disambiguating_delimiter + std::to_string(unique_name_counter++);
}

} // namespace optifol
```

1.16.1.4.3.12 SymbolStandardisingVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Symbol-Standardising Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-12-01
 * @version Development
 */

#ifndef SYMBOLSTANDARDISINGVISITOR_HPP
#define SYMBOLSTANDARDISINGVISITOR_HPP

#include <optional>
#include <unordered_map>
```

```
#include "../../../../IR/MutableVariants/Terms/IMutableTerm.hpp"
#include "../../Terms/ScopedTermResolutionVisitor.hpp"
#include "../MutatingSentenceVisitorBase.hpp"

namespace optifol
{

/**
 * @class SymbolStandardisingVisitor
 * @brief The symbol-standardising visitor validates the semantic correctness of variables and standardises
 * the names thereof throughout the model.
 *
 * @details The symbol-standardising visitor recurses through the model, paying particular attention to
 * variables. It performs two overarching tasks: <ol> <li><b>Verifying semantic correctness:</b> enforces the
 * rules surrounding the usage of variables. In particular, variables may only be introduced by way of a
 * quantifier. If a variable is referenced without having been bound by a quantifier in the relevant scope, an
 * exception is thrown. Dually, if a variable is bound by a quantifier having already been defined in the
 * relevant scope, an exception is thrown.</li> <li><b>Standardising naming of adjacent variables:</b> ensures
 * that syntactically and semantically valid sentences discriminate between variables of the same name when
 * used across different scopes. For example, <code>%Ux(P(x)) | %Ex(Q(x))</code> would be suitably rewritten
 * as <code>%Ux(P(x)) | %Ex_0(Q(x_0))</code>, where <code>x_0</code> is the introduced variable.</li>
 *  </ol>
 *
 * @warning Although multiple passes are not required for this symbol-standardising visitor, it does recurse
 * on any produced terms to ensure a full reduction. On extremely deeply nested sentences, this could cause a
 * machine stack overflow.
 */
class SymbolStandardisingVisitor : public MutatingSentenceVisitorBase
{
public:
    [[nodiscard]] std::string_view get_visitor_name() const override;

    /**
     * @brief Recursively applies the symbol-standardisation procedures to the given quantified sentence root.
     * @param node The root quantified sentence node
     * @throws SemanticException
     */
    void visit(MutableQuantified &node) override;

    /**
     * @brief Recursively applies the symbol-standardisation procedures to the given predicate root node.
     * @param node The root predicate sentence node
     * @throws SemanticException
     */
    void visit(MutablePredicate &node) override;

private:
    static const char *visitor_name;

    /**
     * @brief Name-rewriting rules for variables encountered in the current scope.
     * @details Any variable with a name matching a key of the map should be completely replaced by the
     * variable unique pointer in the corresponding value. Only variables with disambiguated names occupy
     * entries in the map, and the map should be cleared down when the scope is released.
     */
    std::unordered_map<std::string, std::unique_ptr<IMutableTerm>, StringHash, std::equal_to<>>
            rewriting_rules;

    /**
     * @brief The set of pre-disambiguated names bound in the current scope.
     */
    std::unordered_set<std::string> scope;

    /**
     * @brief The set of pre-disambiguated names bound by adjacent scopes, not including the current scope.
     */
    std::unordered_set<std::string> adjacent;

    /**
     * @brief The suffix of the next variable name requiring disambiguation.
     */
    unsigned int unique_name_counter = 0;

    /**
     * @brief The nested term visitor used to assist rewriting of variables nested in terms that are not
     * accessible through the sentence interface (i.e. functions).
     */
    ScopedTermResolutionVisitor term_visitor{scope, rewriting_rules};

    /**
     * @brief Establishes a new scope, introducing the variable bound by the given quantifier. If necessary,
```

```
     * the bound name is disambiguated, in which case a rewriting rule entry is added.
     * @param node The bounding quantifier responsible for the opened scope
     * @return An observing reference for the rule of the opened scope.
     */
    std::optional<decltype(rewriting_rules)::iterator> open_scope(MutableQuantified &node);

    /**
     * @brief Closes the current scope, clearing applicable entries from the scope naming set and the
     * rewriting rule map. The bound variable of the closed scope is comitted to the adjacents naming set.
     * @pre The scope naming set must contain a variable name of the given node.
     * @param node The variable bound by the scope
     * @param rule_reference An observing reference to the rule of the closed scope.
     */
    void close_scope(MutableQuantified &node,
            const std::optional<decltype(rewriting_rules)::iterator> &rule_reference);

    /**
     * @brief Suffix the given variable name with a unique identifier, until it does not conflict with any
     * member of the adjacents naming set.
     * @param name The ambiguous name
     * @return The disambiguated name
     */
    std::string generate_name(const std::string &name);
};

} // namespace optifol

#endif
```

1.16.1.4.3.13 UniversalEliminationVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Universal-Elimination Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-25
 * @version Development
 */

#include "UniversalEliminationVisitor.hpp"

#include "../../../../IR/MutableVariants/Sentences/MutableBinaryConnected.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableQuantified.hpp"
#include "../../../../IR/MutableVariants/Sentences/MutableSentenceRoot.hpp"

namespace optifol
{

const char *UniversalEliminationVisitor::visitor_name = "UniversalElimination";

std::string_view UniversalEliminationVisitor::get_visitor_name() const
{
    return visitor_name;
}

void UniversalEliminationVisitor::visit(MutableQuantified &node)
{
    auto borrowed_sentence = node.take_sentence();
    borrowed_sentence->accept(*this);
    if (extracted_sentence == nullptr)
        node.put_sentence(std::move(borrowed_sentence));
    else
        node.put_sentence(std::move(extracted_sentence));

    if (node.get_quantifier_type() == QuantifierTypes::Universal)
        extracted_sentence = std::move(node.take_sentence());
}

void UniversalEliminationVisitor::visit(MutableBinaryConnected &node)
{
    auto borrowed_operand = node.take_lhs_operand();
    borrowed_operand->accept(*this);
    if (extracted_sentence == nullptr)
        node.put_lhs_operand(std::move(borrowed_operand));
```

```
    else
        node.put_lhs_operand(std::move(extracted_sentence));

    borrowed_operand = node.take_rhs_operand();
    borrowed_operand->accept(*this);
    if (extracted_sentence == nullptr)
        node.put_rhs_operand(std::move(borrowed_operand));
    else
        node.put_rhs_operand(std::move(extracted_sentence));
}

void UniversalEliminationVisitor::visit(MutableSentenceRoot &node)
{
    auto borrowed_sentence = node.take_sentence();
    borrowed_sentence->accept(*this);
    if (extracted_sentence == nullptr)
        node.put_sentence(std::move(borrowed_sentence));
    else
        node.put_sentence(std::move(extracted_sentence));
}

} // namespace optifol
```

1.16.1.4.3.14 UniversalEliminationVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Universal-Elimination Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-25
 * @version Development
 */

#ifndef UNIVERSALELIMINATIONVISITOR_HPP
#define UNIVERSALELIMINATIONVISITOR_HPP

#include <memory>

#include "../../../../IR/MutableVariants/Sentences/IMutableSentence.hpp"
#include "../MutatingSentenceVisitorBase.hpp"

namespace optifol
{

class IMutableSentence;

class UniversalEliminationVisitor : public MutatingSentenceVisitorBase
{
public:
    [[nodiscard]] std::string_view get_visitor_name() const override;

    void visit(MutableQuantified &node) override;

    void visit(MutableBinaryConnected &node) override;

    void visit(MutableSentenceRoot &node) override;

private:
    static const char *visitor_name;

    std::unique_ptr<IMutableSentence> extracted_sentence;
};

} // namespace optifol

#endif
```

1.16.1.5 Terms

1.16.1.5.1 MutatingTermVisitorBase.cpp

```
/*
```

```
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Mutating Term Visitor base class.
 * @author Oliver Dixon
 * @date 2024-11-30
 * @version Development
 */

#include "MutatingTermVisitorBase.hpp"
#include "../../../IR/MutableVariants/Terms/MutableFunction.hpp"

namespace optifol
{

void MutatingTermVisitorBase::visit(MutableFunction &node)
{
    const auto &arguments = node.observe_arguments();

    for (const auto &argument: arguments)
        argument->accept(*this);
}

void MutatingTermVisitorBase::visit(MutableVariable &node)
{
    std::ignore = node;
}

void MutatingTermVisitorBase::visit(MutableSkolemFunction &node)
{
    std::ignore = node;
}

} // namespace optifol
```

1.16.1.5.2 MutatingTermVisitorBase.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Mutating Term Visitor base class.
 * @author Oliver Dixon
 * @date 2024-11-30
 * @version Development
 */

#ifndef MUTATINGTERMVISITORBASE_HPP
#define MUTATINGTERMVISITORBASE_HPP

namespace optifol
{

class MutableFunction;
class MutableVariable;
class MutableSkolemFunction;

class MutatingTermVisitorBase
{
public:
    virtual ~MutatingTermVisitorBase() = default;

    virtual void visit(MutableFunction &node);

    virtual void visit(MutableVariable &node);

    virtual void visit(MutableSkolemFunction &node);
};

} // namespace optifol

#endif
```

1.16.1.5.3 ScopedTermResolutionVisitor.cpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Scoped Term-Resolution Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2025-05-08
 * @version Development
 */

#include "ScopedTermResolutionVisitor.hpp"

#include "../../../Exceptions/SemanticException.hpp"
#include "../../../IR/MutableVariants/Terms/MutableVariable.hpp"

namespace optifol
{

ScopedTermResolutionVisitor::ScopedTermResolutionVisitor(const std::unordered_set<std::string> &scope_hook,
        const std::unordered_map<std::string, std::unique_ptr<IMutableTerm>, StringHash, std::equal_to<>>
                &rewriting_rules_hook) :
    TermResolutionVisitor(rewriting_rules_hook),
    scope_hook(scope_hook)
{
}

void ScopedTermResolutionVisitor::visit(MutableFunction &node)
{
    TermResolutionVisitor::visit(node);
}

void ScopedTermResolutionVisitor::visit(MutableVariable &node)
{
    const auto &name = node.to_string();

    if (!scope_hook.contains(name))
        throw SemanticException("Referenced variable \"" + name + "\" is not defined in the current scope.");
}

} // namespace optifol
```

1.16.1.5.4 ScopedTermResolutionVisitor.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Scoped Term-Resolution Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2025-05-08
 * @version Development
 */

#ifndef SCOPEDTERMRESOLUTIONVISITOR_HPP
#define SCOPEDTERMRESOLUTIONVISITOR_HPP

#include <unordered_set>
#include "TermResolutionVisitor.hpp"

namespace optifol
{

class ScopedTermResolutionVisitor : public TermResolutionVisitor
{
public:
    ScopedTermResolutionVisitor(const std::unordered_set<std::string> &scope_hook,
            const std::unordered_map<std::string, std::unique_ptr<IMutableTerm>, StringHash, std::equal_to<>>
                    &rewriting_rules_hook);

    void visit(MutableFunction &node) override;
```

```
    void visit(MutableVariable &node) override;

private:
    const std::unordered_set<std::string> &scope_hook;
};

} // namespace optifol

#endif
```

1.16.1.5.5 TermResolutionVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the Term-Resolution Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-30
 * @version Development
 */

#include "TermResolutionVisitor.hpp"
#include "../../../IR/MutableVariants/Terms/MutableFunction.hpp"

namespace optifol
{

TermResolutionVisitor::TermResolutionVisitor(
        const std::unordered_map<std::string, std::unique_ptr<IMutableTerm>, StringHash, std::equal_to<>>
                &rewriting_rules_hook) :
    rewriting_rules_hook(rewriting_rules_hook)
{
}

void TermResolutionVisitor::visit(MutableFunction &node)
{
    auto &args = node.observe_arguments();
    const auto argument_count = args.size();

    for (std::remove_const_t<decltype(argument_count)> i = 0; i < argument_count; ++i) {
        const auto &rule = rewriting_rules_hook.find(args[i]->get_disambiguated_name());
        if (rule != rewriting_rules_hook.end() && node != *rule->second)
            /*
             * Rewrite the argument according to the rule if and only if:
             *
             *  - A suitable rule is available, such that the disambiguated name of the argument has been
             * identified as rewritable; and
             *  - The rewriting would change the argument. If this check is not done, an infinite loop would
             * be caused by the following accept call.
             */
            args[i] = rule->second->clone();

        args[i]->accept(*this);
    }
}

} // namespace optifol
```

1.16.1.5.6 TermResolutionVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the Term-Resolution Visitor and its associated rule set.
 * @author Oliver Dixon
 * @date 2024-11-30
 * @version Development
 */
```

```
#ifndef TERMRESOLUTIONVISITOR_HPP
#define TERMRESOLUTIONVISITOR_HPP

#include <string>
#include <unordered_map>

#include "../../../IR/MutableVariants/Terms/IMutableTerm.hpp"
#include "MutatingTermVisitorBase.hpp"

namespace optifol
{

class MutableVariable;

/**
 * @class TermResolutionVisitor
 * @brief The TermResolutionVisitor rewrites a term-based model according to a variable set of substitution
 * rules.
 */
class TermResolutionVisitor : public MutatingTermVisitorBase
{
public:
    /**
     * @brief Construct a new TermResolutionVisitor with a weak observing reference to the substitution rules
     * @param rewriting_rules_hook A weak reference, guaranteed to persist for the lifetime of the
     * TermResolutionVisitor object, mapping term names to replacement terms. Keys are views and have the same
     * lifetime guarantees as the map.
     */
    explicit TermResolutionVisitor(
            const std::unordered_map<std::string, std::unique_ptr<IMutableTerm>, StringHash, std::equal_to<>>
                    &rewriting_rules_hook);

    /**
     * @brief Rewrite a function and all arguments according to the substitution ruleset.
     * @param node The function node to rewrite
     * @details Given an @f$ N @f$-ary function @f$ f \left( x_1, \ldots, x_N \right) @f$, the visitor
     * rewrites the model to produce @f$ f^\prime \left( x_1^\prime, \ldots, x_N^\prime \right) @f$ over the
     * ruleset @f$ V @f$ where
     *  @f[
     *      \alpha =
     *      \begin{cases}
     *          \alpha^\prime & \text{if } \left\{ \alpha \mapsto \alpha^\prime \right\} \in V \text{; or} \\
     *          \alpha & \text{otherwise}
     *      \end{cases}
     *  @f]
     *  for @f$ \alpha \in \left\{ f, x_1, \ldots, x_N \right\} @f$.
     */
    void visit(MutableFunction &node) override;

private:
    const std::unordered_map<std::string, std::unique_ptr<IMutableTerm>, StringHash, std::equal_to<>>
            &rewriting_rules_hook;
};

} // namespace optifol

#endif
```

1.16.2 RegularTargets

1.16.2.1 FeatureBuildingVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the feature-building visitor
 * @author Oliver Dixon
 * @date 2026-02-04
 * @version Development
 */

#include "FeatureBuildingVisitor.hpp"

#include "../../IR/Sentences/Clause.hpp"
#include "../../IR/Terms/Function.hpp"
```

```
namespace optifol
{

FeatureBuildingVisitor::FeatureBuildingVisitor()
{
    features.reserve(Feature::feature_types.size());
    for (const auto feature_type: Feature::feature_types)
        features.emplace_back(feature_type);
}

void FeatureBuildingVisitor::visit(const Clause *const clause) noexcept
{
    reset_counters();

    seen_functions.clear();
    seen_variables.clear();

    for (const auto literal: *clause)
        literal->accept(*this);
}

void FeatureBuildingVisitor::visit(const Literal *const literal) noexcept
{
    enter();
    Feature::get(features, Feature::FeatureType::LiteralCount).increment();

    const auto &args = literal->observe_arguments();
    for (const auto argument: args)
        argument->accept(*this);

    exit();
}

void FeatureBuildingVisitor::visit(const Function *const function) noexcept
{
    enter();

    const auto [it, was_new] = seen_functions.insert(function);
    if (was_new) {
        Feature::get(features, Feature::FeatureType::FunctionCount).increment();
        const auto &args = function->observe_arguments();
        for (const auto argument: args)
            argument->accept(*this);
    }

    exit();
}

void FeatureBuildingVisitor::visit(const SkolemFunction *const skolem_function) noexcept
{
    std::ignore = *skolem_function;

    // Special case: Skolems should not contribute toward the function-count feature, as they are always
    // synthesised.
    enter();
    exit();
}

void FeatureBuildingVisitor::visit(const Variable *const variable) noexcept
{
    enter();

    const auto [it, was_new] = seen_variables.insert(variable);
    if (was_new)
        Feature::get(features, Feature::FeatureType::VariableCount).increment();

    exit();
}

std::vector<Feature> FeatureBuildingVisitor::extract_sorted_vector()
{
    auto saved_features = features;
    reset_counters();
    return saved_features;
}

void FeatureBuildingVisitor::enter() noexcept
{
    Feature::get(features, Feature::FeatureType::MaxDepth).maximise(++current_depth);
}
```

```
void FeatureBuildingVisitor::exit() noexcept
{
    --current_depth;
}

void FeatureBuildingVisitor::reset_counters() noexcept
{
    current_depth = 0;

    for (auto &feature: features)
        feature.reset();
}

} // namespace optifol
```

1.16.2.2 FeatureBuildingVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the feature-building visitor
 * @author Oliver Dixon
 * @date 2026-02-04
 * @version Development
 */

#ifndef OPTIFOL_FEATUREBUILDINGVISITOR_HPP
#define OPTIFOL_FEATUREBUILDINGVISITOR_HPP

#include <vector>

#include "../../IR/Terms/Function.hpp"
#include "../../IR/Terms/Variable.hpp"
#include "../../Inference/Feature.hpp"
#include "../../Optifol.hpp"

namespace optifol
{

class Clause;
class Literal;
class SkolemFunction;

/**
 * @class FeatureBuildingVisitor
 * @brief The FeatureBuildingVisitor recurses over a Clause structure and builds up features according to the
 * state of the immutable AST.
 */
class FeatureBuildingVisitor
{
public:
    FeatureBuildingVisitor();

    void visit(const Clause *clause) noexcept;

    void visit(const Literal *literal) noexcept;

    void visit(const Function *function) noexcept;

    void visit(const SkolemFunction *skolem_function) noexcept;

    void visit(const Variable *variable) noexcept;

    /**
     * @brief Extract the features computed since the previous invocation on the Clause.
     * @return The stored features, in the order specified by the Feature comparator.
     */
    std::vector<Feature> extract_sorted_vector();

private:
    void enter() noexcept;

    void exit() noexcept;
```

```
    void reset_counters() noexcept;

    unsigned int current_depth = 0;

    std::vector<Feature> features;

    RawUnorderedSet<const Function> seen_functions;
    RawUnorderedSet<const Variable> seen_variables;
};

} // namespace optifol

#endif // OPTIFOL_FEATUREBUILDINGVISITOR_HPP
```

1.16.2.3 IObservingBinaryVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Interface specification for the FOL binary visitor
 * @author Oliver Dixon
 * @date 2026-02-02
 * @version Development
 */

#ifndef OPTIFOL_IOBSERVINGBINARYVISITOR_HPP
#define OPTIFOL_IOBSERVINGBINARYVISITOR_HPP

namespace optifol
{

class Literal;
class Variable;
class Function;

/**
 * @class IObservingBinaryVisitor
 * @brief Interface for observing visitors to despatch on two objects.
 */
class IObservingBinaryVisitor
{
public:
    virtual ~IObservingBinaryVisitor() = default;

    // Literal

    [[nodiscard]] virtual bool visit(const Literal &predicate_lhs, const Literal &predicate_rhs) = 0;

    // Variable on LHS

    [[nodiscard]] virtual bool visit(const Variable &variable_lhs, const Function &function_rhs) = 0;

    [[nodiscard]] virtual bool visit(const Variable &variable_lhs, const Variable &variable_rhs) = 0;

    // Function on LHS

    [[nodiscard]] virtual bool visit(const Function &function_lhs, const Function &function_rhs) = 0;

    [[nodiscard]] virtual bool visit(const Function &function_lhs, const Variable &variable_rhs) = 0;
};

} // namespace optifol

#endif // OPTIFOL_IOBSERVINGBINARYVISITOR_HPP
```

1.16.2.4 Unification

1.16.2.4.1 BidirectionalUnificationVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */
```

```cpp
/**
 * @file
 * @brief Class implementation for the FOL binary-unification visitor (bidirectional)
 * @author Oliver Dixon
 * @date 2025-06-08
 * @version Development
 */

#include "BidirectionalUnificationVisitor.hpp"

#include "../../../IR/SymbolRepository.hpp"
#include "../../../IR/Terms/ITerm.hpp"
#include "../../../IR/Terms/Variable.hpp"
#include "UnificationApplicationVisitor.hpp"

namespace optifol
{

BidirectionalUnificationVisitor::BidirectionalUnificationVisitor(
        std::shared_ptr<SymbolRepository> symbol_repository) :
    UnificationVisitor(std::move(symbol_repository)),
    application_visitor(substitutions, this->symbol_repository)
{
}

bool BidirectionalUnificationVisitor::visit(const Function &function_lhs, const Variable &variable_rhs)
{
    return UnificationVisitor::visit(variable_rhs, function_lhs);
}

bool BidirectionalUnificationVisitor::variable_generic(
        const Variable &variable_lhs, const IProcessedTerm &generic_term_rhs)
{
    if (variable_lhs.hash() == generic_term_rhs.hash())
        // If atomics (e.g. variables) are trivially identical, they can be unified without an explicit
        // substitution.
        return true;

    /*
     * If the given LHS variable already has a binding, ensure that its bound mapping can be unified with the
     * other variable.
     */
    const auto &lhs_binding_it = substitutions->unifier.find(variable_lhs);
    if (lhs_binding_it != substitutions->unifier.cend())
        return lhs_binding_it->second->accept(*this, generic_term_rhs);

    // Perform an 'occurs check' only when considering binding a variable to a generic, non-variable term.
    if (occurs_check(variable_lhs, generic_term_rhs))
        return false;

    // If all checks pass, we can do a unification between the variables. Register the replacement and
    // indicate success.
    register_substitution(variable_lhs, generic_term_rhs);
    return true;
}

bool BidirectionalUnificationVisitor::occurs_check(
        const Variable &variable_lhs, const IProcessedTerm &generic_term_rhs)
{
    const auto applied_term = generic_term_rhs.accept(application_visitor);

    const bool is_self_nested = *applied_term == generic_term_rhs
            ? generic_term_rhs.is_self_nested(variable_lhs)
            : applied_term->is_self_nested(variable_lhs);

    application_visitor.discard_new_symbols();
    return is_self_nested;
}

} // namespace optifol
```

1.16.2.4.2 BidirectionalUnificationVisitor.hpp

```cpp
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */
```

```cpp
/**
 * @file
 * @brief Class specification for the FOL binary-unification visitor (bidirectional)
 * @author Oliver Dixon
 * @date 2025-06-08
 * @version Development
 */

#ifndef BIDIRECTIONALUNIFICATIONVISITOR_H
#define BIDIRECTIONALUNIFICATIONVISITOR_H

#include "UnificationApplicationVisitor.hpp"
#include "UnificationVisitor.hpp"

namespace optifol
{

class ITerm;
class SymbolRepository;

/**
 * @class BidirectionalUnificationVisitor
 * @brief Bidirectional variant of the UnificationVisitor, permitting substitutions in the generalisation and
 * the instance, and includes an occurs-check for sanity-checking substitutions.
 */
class BidirectionalUnificationVisitor : public UnificationVisitor
{
public:
    /**
     * @brief Initialise the BidirectionalUnificationVisitor for use with symbols registered in the given
     *  SymbolRepository.
     * @param symbol_repository The SymbolRepository containing symbols for the terms received by the visitor.
     */
    explicit BidirectionalUnificationVisitor(std::shared_ptr<SymbolRepository> symbol_repository);

    [[nodiscard]] bool visit(const Function &function_lhs, const Variable &variable_rhs) override;

protected:
    /**
     * @brief Attempt to unify a variable with a non-variable ("generic") term.
     * @details
     *      To unify a variable with a generic term, they must be one of the following. Providing that
     * unification is successful in the non-trivial sense, a substitution is added to the map. <ul>
     *          <li>Hash-identical: if they have the same hash, they are assumed to refer the same object.
     * Unification is valid in the trivial sense, and an explicit substitution does not need to be
     * recorded.</li> <li>Not chain-ununifiable: if the variable is already bound to a substitution, the bound
     * term must be unifiable to the generic term.</li>
     *      </ul>
     * @param variable_lhs The LHS variable
     * @param generic_term_rhs The RHS generic term
     * @return Can the variable and term be unified?
     */
    [[nodiscard]] bool variable_generic(
            const Variable &variable_lhs, const IProcessedTerm &generic_term_rhs) override;

private:
    /**
     * @brief Determines whether the LHS Variable occurs in the RHS IProcessedTerm, or any applicable
     * substitutions thereof.
     * @details
     *  <p>
     *      A failure of the 'occurs check' procedure informs a unifier whether the introduction of a Variable
     *      substitution will cause a cycle with itself, or with an existing substitution in the unifier @f$
     * \theta @f$.
     *  </p>
     *  <p>
     *      If the presently generated unifier @f$ \theta = \left\{ \theta_1, \ldots, \theta_i \right\} @f$ is
     * already composed of @f$ i @f$ substitutions, and the occurs checker is considering the addition of a
     * new substitution
     *      @f$ \theta_{i+1} = \left[ \alpha \mapsto \beta \right] @f$, <code>true</code> is returned if and
     * only if
     *      @f$ \alpha @f$ appears in the expansion of @f$ \beta @f$ or any of the following:
     *      @f[
     *          \text{Sub}\left( \theta_1, \beta \right), \ldots, \text{Sub}\left( \theta_i, \beta \right).
     *      @f]
     *  </p>
     * @param variable_lhs The Variable for which to search in the substituted enumeration of the RHS
     * IProcessedTerm.
     * @param generic_term_rhs The generic IProcessedTerm to explore under substitutions, searching for the
     * Variable.
     * @return Does the Variable appear in the IProcessedTerm, or any isomorphisms (under substitution) using
```

```
     * the presently generated substitutions?
     */
    [[nodiscard]] bool occurs_check(const Variable &variable_lhs, const IProcessedTerm &generic_term_rhs);

    /**
     * @brief A helper substitution applicator for @ref occurs_check.
     * @note This is <code>mutable</code> because it's essentially a throw-away, single-use cache for the
     * occurs check, and we don't want to break API semantics by indicating the occurs-check procedure is
     * non-constant on the visitor object.
     */
    UnificationApplicationVisitor application_visitor;
};

} // namespace optifol

#endif
```

1.16.2.4.3 UnificationApplicationVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the FOL substitution-application visitor
 * @author Oliver Dixon
 * @date 2026-01-21
 * @version Development
 */

#include "UnificationApplicationVisitor.hpp"

#include "../../../IR/Sentences/Literal.hpp"
#include "../../../IR/SymbolRepository.hpp"
#include "../../../IR/Terms/Function.hpp"

// ReSharper disable CppUnusedIncludeDirective - Full definitions required for transparent hashing of sub.
// map.
#include "../../../IR/Terms/Variable.hpp"
// ReSharper restore CppUnusedIncludeDirective

namespace optifol
{

UnificationApplicationVisitor::UnificationApplicationVisitor(
        std::shared_ptr<Unifier> unifier, std::shared_ptr<SymbolRepository> symbol_repository) :
    substitutions(std::move(unifier)),
    existing_symbol_repository(std::move(symbol_repository)),
    new_symbol_repository(std::make_unique<SymbolRepository>())
{
}

const IProcessedTerm *UnificationApplicationVisitor::visit(const Variable &node) const
{
    const auto it = substitutions->unifier.find(node);
    return it == substitutions->unifier.cend() ? existing_symbol_repository->get_symbol_handle(node)
                                               : it->second;
}

const IProcessedTerm *UnificationApplicationVisitor::visit(const Function &node) const
{
    auto transformed_arguments = apply_to_term_vector(node.observe_arguments());

    /*
     * If the unifier could be successfully applied component-wise to the arguments (indicated by the
     * std::optional containing a vector), create the applied Function symbol and add it to the
     * SymbolRepository. Otherwise, provide a handle to the original unmutated Function.
     */
    return transformed_arguments.has_value()
            ? existing_symbol_repository->add_symbol(std::make_unique<Function>(
                      std::string(node.get_disambiguated_name()), std::move(*transformed_arguments)))
            : existing_symbol_repository->get_symbol_handle(node);
}

const Literal *UnificationApplicationVisitor::visit(const Literal &node) const
{
    auto transformed_arguments = apply_to_term_vector(node.observe_arguments());
```

```
    return transformed_arguments.has_value()
            ? existing_symbol_repository->add_symbol<Literal>(
                    std::make_unique<Literal>(std::string(node.get_name()),
                            std::move(*transformed_arguments), !node.is_negative_polarity()))
            : existing_symbol_repository->get_symbol_handle<Literal>(node);
}

void UnificationApplicationVisitor::discard_new_symbols()
{
    new_symbol_repository = std::make_unique<SymbolRepository>();
}

void UnificationApplicationVisitor::keep_new_symbols()
{
    existing_symbol_repository->inherit_repository(std::move(new_symbol_repository));
    discard_new_symbols();
}

std::optional<std::vector<const IProcessedTerm *>> UnificationApplicationVisitor::apply_to_term_vector(
        const std::vector<const IProcessedTerm *> &terms) const
{
    bool changed = false;
    std::vector<const IProcessedTerm *> transformed_arguments;

    transformed_arguments.reserve(terms.size());

    for (const auto &argument: terms) {
        const auto transformed_argument = argument->accept(*this);
        transformed_arguments.push_back(transformed_argument);

        if (transformed_argument != argument)
            changed = true;
    }

    if (!changed)
        /*
         * It's only worth reporting our transformed argument vector if a transformation occurred on at least
         * one of the arguments. Else it's just the orginal argument vector, and we can indicate this with an
         * empty optional.
         */
        return std::nullopt;

    return transformed_arguments;
}

} // namespace optifol
```

1.16.2.4.4 UnificationApplicationVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the FOL substitution-application visitor
 * @author Oliver Dixon
 * @date 2026-01-21
 * @version Development
 */

#ifndef OPTIFOL_UNIFICATIONAPPLICATIONVISITOR_HPP
#define OPTIFOL_UNIFICATIONAPPLICATIONVISITOR_HPP

#include <optional>
#include <vector>

#include "../../../Inference/Unifier.hpp"

namespace optifol
{

class SymbolRepository;

class Function;
class BinaryConnected;
class Literal;
class SentenceRoot;
```

```
class UnificationApplicationVisitor
{
public:
    explicit UnificationApplicationVisitor(
            std::shared_ptr<Unifier> unifier, std::shared_ptr<SymbolRepository> symbol_repository);

    [[nodiscard]] const IProcessedTerm *visit(const Variable &node) const;

    [[nodiscard]] const IProcessedTerm *visit(const Function &node) const;

    [[nodiscard]] const Literal *visit(const Literal &node) const;

    void discard_new_symbols();

    void keep_new_symbols();

private:
    [[nodiscard]] std::optional<std::vector<const IProcessedTerm *>> apply_to_term_vector(
            const std::vector<const IProcessedTerm *> &terms) const;

    std::shared_ptr<Unifier> substitutions;

    const std::shared_ptr<SymbolRepository> existing_symbol_repository;
    std::unique_ptr<SymbolRepository> new_symbol_repository;
};

} // namespace optifol

#endif // OPTIFOL_UNIFICATIONAPPLICATIONVISITOR_HPP
```

1.16.2.4.5 UnificationVisitor.cpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class implementation for the FOL binary-unification visitor (unidirectional)
 * @author Oliver Dixon
 * @date 2025-06-08
 * @version Development
 */

#include "UnificationVisitor.hpp"

#include <algorithm>
#include <ranges>

#include "../../../IR/Sentences/Literal.hpp"
#include "../../../IR/SymbolRepository.hpp"
#include "../../../IR/Terms/Function.hpp"
#include "../Exceptions/SemanticException.hpp"

namespace optifol
{

UnificationVisitor::UnificationVisitor(std::shared_ptr<SymbolRepository> symbol_repository) :
    symbol_repository(std::move(symbol_repository)),
    substitutions(std::make_shared<Unifier>())
{
}

bool UnificationVisitor::visit(const Literal &predicate_gen, const Literal &predicate_inst)
{
    if (predicate_gen.is_negative_polarity() != predicate_inst.is_negative_polarity())
        return false;

    if (predicate_gen.get_name() != predicate_inst.get_name())
        return false;

    const auto &gen_args = predicate_gen.observe_arguments();
    const auto &inst_args = predicate_inst.observe_arguments();

    if (gen_args.size() != inst_args.size())
        return false;
```

```
    return std::ranges::all_of(std::ranges::views::zip(gen_args, inst_args), [this](const auto &arg_pair)
            { return std::get<0>(arg_pair)->accept(*this, *std::get<1>(arg_pair)); });
}

bool UnificationVisitor::visit(const Variable &variable_gen, const Function &function_inst)
{
    return variable_generic(variable_gen, function_inst);
}

bool UnificationVisitor::visit(const Variable &variable_gen, const Variable &variable_inst)
{
    return variable_generic(variable_gen, variable_inst);
}

bool UnificationVisitor::visit(const Function &function_gen, const Variable &variable_inst)
{
    std::ignore = function_gen;
    std::ignore = variable_inst;

    return false;
}

bool UnificationVisitor::visit(const Function &function_gen, const Function &function_inst)
{
    if (function_gen.get_disambiguated_name() != function_inst.get_disambiguated_name())
        return false;

    const auto &gen_args = function_gen.observe_arguments();
    const auto &inst_args = function_inst.observe_arguments();

    if (gen_args.size() != inst_args.size())
        return false;

    return std::ranges::all_of(std::ranges::views::zip(gen_args, inst_args), [this](const auto &arg_pair)
            { return std::get<0>(arg_pair)->accept(*this, *std::get<1>(arg_pair)); });
}

const Unifier *UnificationVisitor::observe_substitutions() const noexcept
{
    return substitutions.get();
}

std::shared_ptr<Unifier> UnificationVisitor::share_substitutions() const noexcept
{
    return substitutions;
}

void UnificationVisitor::reset_substitutions() const noexcept
{
    substitutions->unifier.clear();
}

void UnificationVisitor::register_substitution(
        const Variable &bound_key, const IProcessedTerm &bound_value) const
{
    const auto variable_repo_ptr = symbol_repository->get_symbol_handle(bound_key);
    const auto bound_repo_ptr = symbol_repository->get_symbol_handle(bound_value);

    if (variable_repo_ptr == nullptr || bound_repo_ptr == nullptr)
        throw SemanticException("Attempted to register substitution for " + bound_key.to_string() +
                " but the "
                "Repository is incomplete.");

    substitutions->unifier.emplace(variable_repo_ptr, bound_repo_ptr);
}

bool UnificationVisitor::variable_generic(
        const Variable &variable_gen, const IProcessedTerm &generic_term_inst)
{
    const auto it = substitutions->unifier.find(variable_gen);
    if (it != substitutions->unifier.end())
        return variable_gen.operator==(*it->first);

    register_substitution(variable_gen, generic_term_inst);
    return true;
}

} // namespace optifol
```

1.16.2.4.6 UnificationVisitor.hpp

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

/**
 * @file
 * @brief Class specification for the FOL binary-unification visitor (unidirectional)
 * @author Oliver Dixon
 * @date 2025-06-08
 * @version Development
 */

#ifndef UNIFICATIONVISITOR_H
#define UNIFICATIONVISITOR_H

#include "../IObservingBinaryVisitor.hpp"
#include "UnificationApplicationVisitor.hpp"

namespace optifol
{

class ITerm;
class SymbolRepository;

/**
 * @class UnificationVisitor
 * @brief Provides a visitor for binary-unification of FOL terms and sentences
 * @details
 *  <p>
 *      For details of the Unification Problem and particulars of the canonical software implementation, see
 *      <i>Artificial Intelligence, A Modern Approach</i> by Russell and Norvig. In brief, the algorithm
 * considers two sentences @f$p@f$ and @f$q@f$ and returns a variable substitution map @f$\theta@f$, or
 * <i>unifier</i> if one exists:
 *      @f[
 *          \text{Unify}(p, q) \mathrel{\vcenter{:}}= \theta \text{ such that }
 *          \text{Sub}(\theta, p) = \text{Sub}(\theta, q).
 *      @f]
 *      For brevity, the definition of our sameness metric is not elaborated here.
 *  </p>
 *  <p>
 *      The most basic UnificationVisitor is unidirectional; that is, it accepts a "generalisation" and an
 * "instance", and substitutions may only be made in the generalisation. If substitutions should also be made
 * in the instance, use the BidirectionalUnificationVisitor instead, which includes an occurs-check to verify
 * correctness of substitutions.
 *  </p>
 *  <p>
 *      The overall association rules within the constraints of the Optifol type systems is:
 *      <ul>
 *          <li>Predicates might unify with predicates only;</li>
 *          <li>Variables might unify with any term;</li>
 *          <li>Functions might unify with functions only; and</li>
 *          <li>Any term might unify with variables only.</li>
 *      </ul>
 *      These rules are reflected in the visitor member functions, and must be respected by unifiable types
 * implementing the acceptor member functions. Note that triple-despatch may be necessary in some cases
 * (acceptor calling another overload acceptor, which then calls the visitor) to correctly explore
 * term-specialised cases.
 *  </p>
 *  <p>
 *      Following a unification attempt (<code>visit</code> call), any applicable substitutions will have been
 * traced by the visitor instance and written to an internal state accessible with @ref observe_substitutions.
 * If unification was successful, indicated by the return code of <code>visit</code>, the set consists of
 * most-general unifiers. If unification was unsuccessful, the substitutions will not necessarily produce
 * matching clauses under application, and the state should be reset with @ref reset_substitutions. This is a
 * conscious API design choice, as it is occasionally useful for users to inspect the partial substitution
 * trace of a failed unification.
 *  </p>
 * @note
 *  <p>
 *      There is opportunity for significant optimisation of this procedure. In particular, the implementation
 *      here is reminiscent of Russell and Norvig's adaptation of J. A. Robinson's 1965 exponential algorithm.
 * At some point, this visitor should use the algorithm of de Champeaux (2022) which is linear-bounded. A Java
 * reference implementation can be found at https://github.com/ddccc/Unification. See also the paper:
 *      https://doi.org/10.1007/s10817-022-09635-1.
 *  </p>
 * @see BidirectionalUnificationVisitor for the unification visitor to produce substitutions on both the
 * generalisation and the instance.
```

```
 * @see UnificationApplicationVisitor for the mutating visitor to apply Unifier elements to ASTs.
 */
class UnificationVisitor : public IObservingBinaryVisitor
{
public:
    /**
     * @brief Initialise the UnificationVisitor for use with symbols registered in the given SymbolRepository.
     * @param symbol_repository The SymbolRepository containing symbols for the terms received by the visitor.
     */
    explicit UnificationVisitor(std::shared_ptr<SymbolRepository> symbol_repository);

    /**
     * @brief Attempt to unify two predicates (Literals) on the predicate and component-wise arguments.
     * @details Unification between two predicates will succeed if and only if they are identically named,
     * have the same number of term arguments, have the same polarity, and the terms can be zipped and
     * mutually unified i.a.w. other rules described herein.
     * @param predicate_gen The generalisation Literal to unify
     * @param predicate_inst The instance Literal to unify
     * @return Can the generalisation be unified into the instance?
     */
    [[nodiscard]] bool visit(const Literal &predicate_gen, const Literal &predicate_inst) override;

    /**
     * @brief Attempt to unify a Variable and a Function term.
     * @param variable_gen The LHS Variable to unify
     * @param function_inst The RHS Function to unify
     * @return Can the LHS and RHS objects be unified?
     */
    [[nodiscard]] bool visit(const Variable &variable_gen, const Function &function_inst) override;

    /**
     * @brief Attempt to unify two Variable terms.
     * @details Unification between two Variable terms will succeed according to the same conditions as those
     * required by @ref visit(const Variable&, const Constant&).
     * @param variable_gen The LHS Variable to unify
     * @param variable_inst The RHS Variable to unify
     * @return Can the LHS and RHS Variable objects be unified?
     */
    [[nodiscard]] bool visit(const Variable &variable_gen, const Variable &variable_inst) override;

    /**
     * @brief Attempt to unify a Function and a Variable.
     * @param function_gen The LHS Function to unify
     * @param variable_inst The RHS Variable to unify
     * @return False, since a Function and a Variable could only be unified if the Variable could be
     * substituted. In this case, it's in the instance.
     */
    [[nodiscard]] bool visit(const Function &function_gen, const Variable &variable_inst) override;

    /**
     * @brief Attempt to unify two Function terms.
     * @details Unification between two Function terms will succeed according to similar conditions as those
     * required by @ref visit(const Literal&, const Literal&), i.e. identically named symbols and
     * component-wise unification.
     * @param function_gen The LHS Function to unify
     * @param function_inst The RHS Function to unify
     * @return Can the LHS and RHS Function objects be unified?
     */
    [[nodiscard]] bool visit(const Function &function_gen, const Function &function_inst) override;

    /**
     * @brief Observe the working set of substitutions produced since the last @ref reset_substitutions call.
     *
     * @details The Unifier set consists of substitutions explored by the visitor whilst attempting to force
     * two clauses to be equivalent under application. If a <code>visit</code> call indicated successful
     * unification, the substitutions represent the most-general unifiers required to make the two respective
     * clauses equivalent under the UnificationApplicationVisitor. An empty set, or set returned following a
     * failed unification attempt, may be useful to callers but are not unifiers.
     *
     * @return The working Unifier set.
     */
    [[nodiscard]] const Unifier *observe_substitutions() const noexcept;

    [[nodiscard]] std::shared_ptr<Unifier> share_substitutions() const noexcept;

    /**
     * @brief Clear unifying substitutions and reset the state such that @ref observe_substitutions produces
     * an empty working set.
     */
    void reset_substitutions() const noexcept;
```

```
protected:
    /**
     * @brief Attempt to unify a variable with a generic term.
     * @details To unify a variable with a generic term in the unidirectional case, the mapping must either be
     * new, or be identical to an existing entry.
     * @param variable_gen The LHS variable
     * @param generic_term_inst The RHS generic term
     * @return Can the variable and term be unified?
     */
    [[nodiscard]] virtual bool variable_generic(
            const Variable &variable_gen, const IProcessedTerm &generic_term_inst);

    /**
     * @brief Registers a new substitution @f$ \alpha \mapsto \beta @f$ for a Variable @f$ \alpha @f$ and
     * generic IProcessedTerm @f$ \beta @f$ in terms of their existing pointers in the @ref symbol_repository.
     * @param bound_key The @f$ \alpha @f$ Variable key to bind.
     * @param bound_value The @f$ \beta @f$ IProcessedTerm binding.
     * @throws SemanticException if either @f$ \alpha @f$ or @f$ \beta @f$ do not exist in the @ref
     * symbol_repository.
     */
    void register_substitution(const Variable &bound_key, const IProcessedTerm &bound_value) const;

    /**
     * @brief The SymbolRepository for the environment of the unified pair, provided by the consumer.
     */
    const std::shared_ptr<SymbolRepository> symbol_repository;

    /**
     * @brief The working set of Variable-to-Term substitutions for the unification attempt. Once unification
     * has returned a verdict, the final set can be observed with @ref observe_substitutions.
     */
    const std::shared_ptr<Unifier> substitutions;
};

} // namespace optifol

#endif
```

2 CMakeLists.txt

```
# Copyright (c) All Rights Reserved
# 2025 Oliver Dixon <od641@york.ac.uk>

# File: CMakeLists.txt
# Brief: Build the Optifol system with CMake
# Author: Oliver Dixon
# Date: 2024-11-15
# Version: Development

cmake_minimum_required(VERSION 3.10 FATAL_ERROR)
include(FetchContent)
project(Optifol)

set(CMAKE_CXX_STANDARD 26)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
find_package(PkgConfig REQUIRED)

# Strip release-like builds, except debuginfo release builds.
function(utils_strip TARGET)
    add_custom_command(
            TARGET "${TARGET}" POST_BUILD
            COMMAND $<$<OR:$<CONFIG:release>,$<CONFIG:minsizerel>>:${CMAKE_STRIP}>
            ARGS --strip-all $<TARGET_FILE:${TARGET}>
    )
endfunction()

########################################
## Optifol Generated Object Library ##
########################################

# The "generated" object library controls the invocation of all non-GTK code-generators required for Optifol.
# Namely: Flex (lexer, generates C++) and Bison (parser, generates C++). All generated source translation
# units are placed in ${GENERATED_SRC_DIR}, and headers are placed in ${GENERATED_INCLUDE_DIR}.

find_package(FLEX REQUIRED)
find_package(BISON REQUIRED)
pkg_check_modules(GTKMM REQUIRED gtkmm-4.0)
```

```
set(GENERATED_INCLUDE_DIR ${CMAKE_CURRENT_BINARY_DIR}/include)
set(GENERATED_SRC_DIR ${GENERATED_INCLUDE_DIR}/generated)
file(MAKE_DIRECTORY ${GENERATED_SRC_DIR})

bison_target(FOLParser
        Source/Interpreter/FOLParser.y
        ${GENERATED_SRC_DIR}/BaseFOLParser.cpp
        COMPILE_FLAGS "-t -Wcounterexamples"
)

flex_target(FOLLexer
        Source/Interpreter/FOLLexer.l
        ${GENERATED_SRC_DIR}/FOLLexer.cpp
)

bison_target(GoogleTestParser
        Source/UserTesting/Execution/PayloadManagement/GoogleTestParser.y
        ${GENERATED_SRC_DIR}/BaseGoogleTestParser.cpp
        COMPILE_FLAGS "-t -Wcounterexamples"
)

flex_target(GoogleTestLexer
        Source/UserTesting/Execution/PayloadManagement/GoogleTestLexer.l
        ${GENERATED_SRC_DIR}/GoogleTestLexer.cpp
)

# Note that a very useful quality example of using CMake to generate multiple subclassed C++ lexers and
# parsers is  OpenSTA from Parallax Software: https://github.com/parallaxsw/OpenSTA.
add_flex_bison_dependency(FOLLexer FOLParser)
add_flex_bison_dependency(GoogleTestLexer GoogleTestParser)

add_library(OptifolGenerated OBJECT
        ${BISON_FOLParser_OUTPUTS}
        ${FLEX_FOLLexer_OUTPUTS}
        ${BISON_GoogleTestParser_OUTPUTS}
        ${FLEX_GoogleTestLexer_OUTPUTS}
)

# Flex and Bison generate all sorts of suspect melting pots of C and C++. It is assumed safe and correct to
# suppress all compiler warnings produced by these programs, hence the grouping of all generated code into a
# distinct CMake target.
target_compile_options(OptifolGenerated PRIVATE -Wno-everything)

target_include_directories(OptifolGenerated PRIVATE
        Source/Interpreter
        Source/UserTesting/Execution/PayloadManagement
        ${GENERATED_INCLUDE_DIR}
)

target_include_directories(OptifolGenerated SYSTEM PRIVATE
        ${GTKMM_INCLUDE_DIRS}
)

set_source_files_properties(
        ${BISON_FOLParser_OUTPUTS}
        ${FLEX_FOLLexer_OUTPUTS}
        ${BISON_GoogleTestParser_OUTPUTS}
        ${FLEX_GoogleTestLexer_OUTPUTS}
        PROPERTIES GENERATED TRUE
)

####################################
## Optifol Common Object Library ##
####################################

add_library(OptifolCommon OBJECT
        Source/Interpreter/FOLLexer.hpp
        Source/IR/MutableVariants/Terms/IMutableTerm.hpp
        Source/IR/MutableVariants/Sentences/IMutableSentence.hpp
        Source/IR/MutableVariants/Terms/MutableVariable.hpp
        Source/IR/MutableVariants/Terms/MutableFunction.hpp
        Source/IR/MutableVariants/Sentences/MutablePredicate.hpp
        Source/IR/MutableVariants/Sentences/MutableBinaryConnected.hpp
        Source/IR/MutableVariants/Sentences/MutableQuantified.hpp
        Source/Interpreter/FOLParser.hpp
        Source/Visitors/MutableTargets/Sentences/CNFNormalisers/ImplicationEliminationVisitor.hpp
        Source/Visitors/MutableTargets/Sentences/MutatingSentenceVisitorBase.hpp
        Source/Visitors/MutableTargets/Sentences/CNFNormalisers/DMLVisitor.hpp
        Source/IR/MutableVariants/Sentences/MutableBinaryConnected.cpp
        Source/IR/MutableVariants/Sentences/MutableQuantified.cpp
        Source/IR/MutableVariants/Sentences/MutablePredicate.cpp
```

Source/Visitors/MutableTargets/Sentences/CNFNormalisers/UniversalEliminationVisitor.hpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/DisjunctionDistributionVisitor.hpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/DisjunctionDistributionVisitor.cpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/UniversalEliminationVisitor.cpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/ImplicationEliminationVisitor.cpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/DMLVisitor.cpp
Source/Visitors/MutableTargets/Observers/TextSerialiserVisitor.cpp
Source/Visitors/MutableTargets/Observers/TextSerialiserVisitor.hpp
Source/Visitors/MutableTargets/Observers/IObservingNodeVisitor.hpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/SymbolStandardisingVisitor.cpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/SymbolStandardisingVisitor.hpp
Source/Exceptions/SemanticException.hpp
Source/Visitors/MutableTargets/Terms/MutatingTermVisitorBase.hpp
Source/Visitors/MutableTargets/Terms/TermResolutionVisitor.cpp
Source/Visitors/MutableTargets/Terms/TermResolutionVisitor.hpp
Source/Visitors/MutableTargets/Terms/MutatingTermVisitorBase.cpp
Source/Visitors/MutableTargets/Sentences/MutatingSentenceVisitorBase.cpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/QuantifierExtractingVisitor.cpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/QuantifierExtractingVisitor.hpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/SkolemIntroducingVisitor.cpp
Source/Visitors/MutableTargets/Sentences/CNFNormalisers/SkolemIntroducingVisitor.hpp
Source/Optifol.hpp
Source/Logging.hpp
Source/Logging.cpp
Source/IR/MutableVariants/Sentences/MutableSentenceRoot.hpp
Source/IR/MutableVariants/Sentences/MutableSentenceRoot.cpp
Source/IR/MutableVariants/Terms/MutableFunction.cpp
Source/IR/MutableVariants/Terms/MutableVariable.cpp
Source/IR/MutableVariants/Terms/MutableSkolemFunction.cpp
Source/IR/MutableVariants/Terms/MutableSkolemFunction.hpp
Source/Visitors/MutableTargets/Terms/ScopedTermResolutionVisitor.cpp
Source/Visitors/MutableTargets/Terms/ScopedTermResolutionVisitor.hpp
Source/Exceptions/ParseError.hpp
Source/IHashable.hpp
Source/Visitors/RegularTargets/Unification/BidirectionalUnificationVisitor.cpp
Source/Visitors/RegularTargets/Unification/BidirectionalUnificationVisitor.hpp
Source/IR/MutableVariants/OwningBuildable.hpp
Source/IR/SymbolRepository.cpp
Source/IR/SymbolRepository.hpp
Source/IR/Sentences/ISentence.hpp
Source/IR/Terms/IProcessedTerm.hpp
Source/IR/Sentences/Literal.cpp
Source/IR/Sentences/Literal.hpp
Source/IR/Terms/Variable.cpp
Source/IR/Terms/Variable.hpp
Source/IR/Terms/Function.cpp
Source/IR/Terms/Function.hpp
Source/IR/Terms/IProcessedTerm.hpp
Source/CompositeSerialisationHelpers.hpp
Source/IR/Sentences/BinaryConnected.cpp
Source/IR/Sentences/BinaryConnected.hpp
Source/Visitors/MutableTargets/RepositoryBuildingVisitor.cpp
Source/Visitors/MutableTargets/RepositoryBuildingVisitor.hpp
Source/IR/Sentences/SentenceRoot.cpp
Source/IR/Sentences/SentenceRoot.hpp
Source/Visitors/MutableTargets/Observers/LaTeXSerialisationVisitor.cpp
Source/Visitors/MutableTargets/Observers/LaTeXSerialisationVisitor.hpp
Source/IR/Sentences/IProcessedSentence.hpp
Source/UserTesting/Execution/PayloadManagement/GoogleTestLexer.hpp
Source/UserTesting/Execution/PayloadManagement/GoogleTestParser.hpp
Source/UserTesting/Execution/TestResult.cpp
Source/UserTesting/Execution/TestResult.hpp
Source/GUI/MainWindow.cpp
Source/GUI/MainWindow.hpp
Source/GUI/Application.cpp
Source/GUI/Application.hpp
Source/GUI/GTKHelpers.hpp
Source/Storage/TreeNode.cpp
Source/Storage/TreeNode.hpp
Source/Storage/Project.cpp
Source/Storage/Project.hpp
Source/Storage/Subsystem.cpp
Source/Storage/Subsystem.hpp
Source/Storage/StorageObjectBase.hpp
Source/GUI/ProjectHierarchyPane/ProjectHierarchyPane.cpp
Source/GUI/ProjectHierarchyPane/ProjectHierarchyPane.hpp
Source/GUI/RequirementsIndexArea/RequirementsIndexArea.cpp
Source/GUI/RequirementsIndexArea/RequirementsIndexArea.hpp
Source/GUI/ContextButtonCorrespondence.cpp
Source/GUI/ContextButtonCorrespondence.hpp
Source/Storage/Requirement.cpp

Source/Storage/Requirement.hpp
Source/Storage/StorageObjectBase.cpp
Source/GUI/AnalysisArea/AnalysisArea.cpp
Source/GUI/AnalysisArea/AnalysisArea.hpp
Source/Storage/AnalysisGroup.cpp
Source/Storage/AnalysisGroup.hpp
Source/GUI/ReportsArea/ReportsArea.cpp
Source/GUI/ReportsArea/ReportsArea.hpp
Source/GUI/ReportsArea/ReportsAreaGenerateLaTeXPopover.cpp
Source/GUI/ReportsArea/ReportsAreaGenerateLaTeXPopover.hpp
Source/GUI/TestingArea/TestingArea.cpp
Source/GUI/TestingArea/TestingArea.hpp
Source/GUI/IWindowArea.hpp
Source/GUI/AnalysisArea/AnalysisAreaNewAnalysisGroupPopover.cpp
Source/GUI/AnalysisArea/AnalysisAreaNewAnalysisGroupPopover.hpp
Source/GUI/RequirementsIndexArea/IndexNewRequirementPopover.cpp
Source/GUI/RequirementsIndexArea/IndexNewRequirementPopover.hpp
Source/GUI/RequirementsIndexArea/IndexDuplicateRequirementPopover.cpp
Source/GUI/RequirementsIndexArea/IndexDuplicateRequirementPopover.hpp
Source/GUI/RequirementsIndexArea/IndexDeleteRequirementPopover.cpp
Source/GUI/RequirementsIndexArea/IndexDeleteRequirementPopover.hpp
Source/UserTesting/Execution/PayloadManagement/TestListenerBase.hpp
Source/UserTesting/Execution/PayloadManagement/GoogleTestListener.cpp
Source/UserTesting/Execution/PayloadManagement/GoogleTestListener.hpp
Source/UserTesting/Modelling/Test.cpp
Source/UserTesting/Modelling/Test.hpp
Source/GUI/ProcessExecutor.cpp
Source/GUI/ProcessExecutor.hpp
Source/UserTesting/Modelling/TestGroup.cpp
Source/UserTesting/Modelling/TestGroup.hpp
Source/Storage/ObjectGroup.hpp
Source/GUI/TestingArea/TestingRunTestsPopover.cpp
Source/GUI/TestingArea/TestingRunTestsPopover.hpp
Source/GUI/StreamingProcessExecutor.cpp
Source/GUI/StreamingProcessExecutor.hpp
Source/UserTesting/Execution/PayloadManagement/TestListenerBase.cpp
Source/UserTesting/Execution/TestExecutable.cpp
Source/UserTesting/Execution/TestExecutable.hpp
Source/GUI/RequirementsIndexArea/ManageTestsPopover.cpp
Source/GUI/RequirementsIndexArea/ManageTestsPopover.hpp
Source/UserTesting/Discovery/DiscoveryTestExecutable.cpp
Source/UserTesting/Discovery/DiscoveryTestExecutable.hpp
Source/UserTesting/Discovery/TestSpecificationEntry.cpp
Source/UserTesting/Discovery/TestSpecificationEntry.hpp
Source/UserTesting/Discovery/DiscoveryTestFixture.cpp
Source/UserTesting/Discovery/DiscoveryTestFixture.hpp
Source/UserTesting/Discovery/GoogleTestDiscoveryExecutable.cpp
Source/UserTesting/Discovery/GoogleTestDiscoveryExecutable.hpp
Source/UserTesting/Modelling/ITestModelNode.hpp
Source/UserTesting/Modelling/ExecutionGroup.cpp
Source/UserTesting/Modelling/ExecutionGroup.hpp
Source/UserTesting/Modelling/GoogleExecutionGroup.cpp
Source/UserTesting/Modelling/GoogleExecutionGroup.hpp
Source/Reporting/IReportGenerator.hpp
Source/Reporting/LaTeXReportGenerator.cpp
Source/Reporting/LaTeXReportGenerator.hpp
Source/GUI/TestingArea/TestingNewTestGroupPopover.cpp
Source/GUI/TestingArea/TestingNewTestGroupPopover.hpp
Source/GUI/TestingArea/TestingCopyToTestGroupPopover.cpp
Source/GUI/TestingArea/TestingCopyToTestGroupPopover.hpp
Source/GUI/TestingArea/TestingCopyMovePopoverBase.cpp
Source/GUI/TestingArea/TestingCopyMovePopoverBase.hpp
Source/GUI/TestingArea/TestingMoveToTestGroupPopover.cpp
Source/GUI/TestingArea/TestingMoveToTestGroupPopover.hpp
Source/GUI/TestingArea/TestingDeleteTestGroupPopover.cpp
Source/GUI/TestingArea/TestingDeleteTestGroupPopover.hpp
Source/GUI/TestingArea/TestingRenameTestGroupPopover.cpp
Source/GUI/TestingArea/TestingRenameTestGroupPopover.hpp
Source/GUI/TestingArea/TestingFailedView.cpp
Source/GUI/TestingArea/TestingFailedView.hpp
Source/UserTesting/Execution/PartialTestResult.cpp
Source/UserTesting/Execution/PartialTestResult.hpp
Source/Inference/Prover.cpp
Source/Inference/Prover.hpp
Source/Visitors/RegularTargets/Unification/UnificationApplicationVisitor.cpp
Source/Visitors/RegularTargets/Unification/UnificationApplicationVisitor.hpp
Source/Inference/ExpressionFactory.cpp
Source/Inference/ExpressionFactory.hpp
Source/IR/Sentences/Clause.hpp
Source/IR/Sentences/Clause.cpp
Source/Inference/Unifier.hpp

```
        Source/Inference/Resolvent.cpp
        Source/Inference/Resolvent.hpp
        Source/ISerialisable.hpp
        Source/GUI/AnalysisArea/AnalysisQuery.cpp
        Source/GUI/AnalysisArea/AnalysisQuery.hpp
        Source/GUI/AnalysisArea/AnalysisQueryCanvas.cpp
        Source/GUI/AnalysisArea/AnalysisQueryCanvas.hpp
        Source/Inference/Unifier.cpp
        Source/Inference/QueryResult.hpp
        Source/Inference/ResolventQueue.hpp
        Source/Inference/ResolventQueue.cpp
        Source/Inference/ProofTreeNode.hpp
        Source/IR/Terms/SkolemFunction.cpp
        Source/IR/Terms/SkolemFunction.hpp
        Source/GUI/AnalysisArea/CanvasSupport.hpp
        Source/Visitors/RegularTargets/Unification/UnificationVisitor.cpp
        Source/Visitors/RegularTargets/Unification/UnificationVisitor.hpp
        Source/Visitors/RegularTargets/IObservingBinaryVisitor.hpp
        Source/Visitors/RegularTargets/FeatureBuildingVisitor.cpp
        Source/Visitors/RegularTargets/FeatureBuildingVisitor.hpp
        Source/Inference/FVIKnowledgeBase.cpp
        Source/Inference/FVIKnowledgeBase.hpp
        Source/Inference/Feature.hpp
)

target_include_directories(OptifolCommon PRIVATE
        Source/Interpreter/
        Source/UserTesting/Execution/PayloadManagement
        ${GENERATED_INCLUDE_DIR}
)

target_include_directories(OptifolCommon SYSTEM PRIVATE
        ${GTKMM_INCLUDE_DIRS}
        ${RAPIDJSON_INCLUDE_DIRS}
)

##########################################
## Optifol GUI Application Executable ##
##########################################

find_program(GLIB_COMPILE_RESOURCES NAMES glib-compile-resources REQUIRED)
find_package(log4cxx REQUIRED)
find_package(RapidJSON REQUIRED)

set(RESOURCE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/Resources)
set(RESOURCE_XML_IN ${RESOURCE_DIR}/optifol.gresource.xml.in)
set(RESOURCE_XML ${GENERATED_SRC_DIR}/optifol.gresource.xml)
set(RESOURCE_OUT ${GENERATED_SRC_DIR}/resources.c)
set(RESOURCE_DEP ${GENERATED_SRC_DIR}/optifol.gresource.d)

set(AWK_SUBSTITUTION_SCRIPT ${RESOURCE_DIR}/substitute.awk)

set(MAINWINDOW_RESOURCE_INPUT ${RESOURCE_DIR}/UI/MainWindow/MainWindow.ui.in)
set(MAINWINDOW_RESOURCE_OUTPUT ${GENERATED_SRC_DIR}/MainWindow.ui)
set(MAINWINDOW_RESOURCE_COMPONENTS
        ${RESOURCE_DIR}/UI/MainWindow/ProjectPane/ProjectPane.ui
        ${RESOURCE_DIR}/UI/MainWindow/ProjectPane/NewProjectPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/ProjectPane/NewSubsystemPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/ProjectPane/EditStructurePopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/ProjectPane/DeleteStructurePopover.ui

        ${RESOURCE_DIR}/UI/MainWindow/RequirementsIndexArea/NewRequirementPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/RequirementsIndexArea/DeleteRequirementPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/RequirementsIndexArea/EditRequirementPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/RequirementsIndexArea/DuplicateRequirementPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/RequirementsIndexArea/ManageTestsPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/RequirementsIndexArea/RequirementsIndexView.ui

        ${RESOURCE_DIR}/UI/MainWindow/AnalysisArea/AnalysisView.ui
        ${RESOURCE_DIR}/UI/MainWindow/AnalysisArea/NewAnalysisGroupPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/AnalysisArea/DeleteAnalysisGroupPopover.ui

        ${RESOURCE_DIR}/UI/MainWindow/TestingArea/TestingView.ui
        ${RESOURCE_DIR}/UI/MainWindow/TestingArea/NewTestGroupPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/TestingArea/RenameTestGroupPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/TestingArea/DeleteTestGroupPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/TestingArea/CopyToTestGroupPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/TestingArea/MoveToTestGroupPopover.ui
        ${RESOURCE_DIR}/UI/MainWindow/TestingArea/RunTests.ui

        ${RESOURCE_DIR}/UI/MainWindow/ReportsArea/ReportsArea.ui
```

```
        ${RESOURCE_DIR}/UI/MainWindow/ReportsArea/GenerateLaTeXPopover.ui

        ${RESOURCE_DIR}/UI/MainWindow/replacements.txt
)

# Generate the Main Window UI specification file using the given Awk script. The Awk script nominally takes a
# number of key-value pairs, defined in ./replacements.txt (relative to the location of the UI files), and
# replaces all instances of the key with the contents of the file whose path is detained by the value.
add_custom_command(
        OUTPUT ${MAINWINDOW_RESOURCE_OUTPUT}
        MAIN_DEPENDENCY ${MAINWINDOW_RESOURCE_INPUT}
        DEPENDS ${MAINWINDOW_RESOURCE_COMPONENTS} ${AWK_SUBSTITUTION_SCRIPT}
        WORKING_DIRECTORY ${RESOURCE_DIR}/UI/MainWindow
        COMMAND awk -f ${AWK_SUBSTITUTION_SCRIPT} > ${MAINWINDOW_RESOURCE_OUTPUT}
        ARGS ${MAINWINDOW_RESOURCE_INPUT}
)

# Performs any CMake variable substitutions in the GResources manifest. This is required due to some included
# resource files being generated dynamically, hence existing in an implementation-defined generated source
# directory.
configure_file(
        ${RESOURCE_XML_IN}
        ${RESOURCE_XML}
)

# The below command generates the 'resources.c' pseudo-blob with 'glib-compile-resources', containing octal
# representations of GTK resources. This is later linked into the GUI target only. A Make-style depfile is also
# created and retained in the generated sources root, using paths relative to the current binary directory, or
# absolute paths, as required by the Ninja backend.
add_custom_command(
        OUTPUT ${RESOURCE_OUT}
        MAIN_DEPENDENCY ${RESOURCE_XML}
        DEPENDS ${MAINWINDOW_RESOURCE_OUTPUT}
        DEPFILE ${RESOURCE_DEP}
        VERBATIM
        WORKING_DIRECTORY ${CMAKE_CURRENT_BINARY_DIR} # Force glib-compile-resources to emit relative paths
        BYPRODUCTS ${RESOURCE_DEP} # Don't let Ninja absorb the depfile into its internal database
        COMMAND ${GLIB_COMPILE_RESOURCES}
        ARGS
        --generate-source
        --generate-phony-targets # Individual resource files are not generated
        --dependency-file=${RESOURCE_DEP}
        --target=${RESOURCE_OUT}
        --sourcedir=${RESOURCE_DIR}
        ${RESOURCE_XML}
        COMMAND sed -i "s|${CMAKE_CURRENT_BINARY_DIR}/||g" ${RESOURCE_DEP} # Clean up paths that are absolute
)

# Suppress compiler warnings known to be harmlessly invoked by glib-compile-resources translation units.
set_source_files_properties(
        ${RESOURCE_OUT}
        PROPERTIES COMPILE_OPTIONS
        "-Wno-overlength-strings;-Wno-reserved-macro-identifier;-Wno-unused-macros"
)

add_executable(OptifolGUI
        ${RESOURCE_OUT}
        Source/GUI/GUIDriver.cpp
)

set(COMMON_COMPILER_OPTIONS
        -Weverything
        -Wno-exit-time-destructors
        -Wno-unknown-pragmas # Support some IntelliJ IDEA in-line directives
        -Wno-c++98-compat
        -Wno-padded
        -Wno-c++98-compat-pedantic
        -Wno-switch-default
        -Wno-shadow-field-in-constructor
        -Wno-weak-vtables
        -Wno-shadow-field
        -Wno-global-constructors
        -fsafe-buffer-usage-suggestions
)

target_compile_options(OptifolGUI PRIVATE ${COMMON_COMPILER_OPTIONS})

target_link_libraries(OptifolGUI PRIVATE
        OptifolCommon
        OptifolGenerated
        log4cxx
```

```
        ${GTKMM_LINK_LIBRARIES}
)

target_include_directories(OptifolGUI SYSTEM PRIVATE
        ${GTKMM_INCLUDE_DIRS}
        ${RAPIDJSON_INCLUDE_DIRS}
)

utils_strip(OptifolGUI)

##########################################
## Optifol Google Test Suite Executable ##
##########################################

include(GoogleTest)
cmake_policy(SET CMP0135 NEW)

FetchContent_Declare(
        googletest
        GIT_REPOSITORY https://github.com/google/googletest.git
        GIT_TAG 52eb8108c5bdec04579160ae17225d66034bd723 # v.1.17.0
)

set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
FetchContent_MakeAvailable(googletest)

enable_testing()

add_executable(OptifolTesting
        Source/Tests/FOLParserTest.cpp
        Source/Tests/CNFNormalisationTest.cpp
        Source/Tests/BidirectionalUnificationTest.cpp
        Source/Tests/UnidirectionalUnificationTest.cpp
        Source/Tests/GoogleTestSupport.hpp
        Source/Tests/RepositoryBuildingTest.cpp
        Source/Tests/ResolutionTest.cpp
        Source/Tests/LiteralOrganisationTest.cpp
        Source/Tests/FeatureVectorIndexTest.cpp
)

target_include_directories(OptifolTesting PRIVATE
        Source/Interpreter/
)

target_include_directories(OptifolTesting SYSTEM PRIVATE
        ${GTKMM_INCLUDE_DIRS}
        ${RAPIDJSON_INCLUDE_DIRS}
        ${GENERATED_INCLUDE_DIR}
)

target_link_libraries(OptifolTesting PRIVATE
        OptifolCommon
        OptifolGenerated
        log4cxx
        GTest::gtest_main
        ${GTKMM_LINK_LIBRARIES}
)

target_compile_options(OptifolTesting PRIVATE
        ${COMMON_COMPILER_OPTIONS}
        -Wno-global-constructors # GTest is known to generate declarations which require global constructors
)

gtest_discover_tests(OptifolTesting)

#######################################
## Optifol Documentation Generation ##
#######################################

set(DOXYGEN_OUTPUT_DIR ${CMAKE_CURRENT_SOURCE_DIR}/GeneratedDocumentation)
set(CPPREFERENCE_DOXYGEN_WEB_TAG ${DOXYGEN_OUTPUT_DIR}/cppreference-doxygen-web.tag.xml)
set(DOXYGEN_SENTINEL ${DOXYGEN_OUTPUT_DIR}/.doxygen_cmake_sentinel)

# Collate declared source files from all eligible targets containing Doxygen directives.
set(DOXYGEN_TARGETS OptifolCommon OptifolGenerated OptifolTesting OptifolGUI)
set(DOXYGEN_SOURCES)
foreach (target IN LISTS DOXYGEN_TARGETS)
    get_target_property(target_sources ${target} SOURCES)
    if (target_sources)
        list(APPEND DOXYGEN_SOURCES ${target_sources})
    endif ()
```

```
endforeach ()

# Download a known version of the cppreference.com Doxygen web tag XML file.
add_custom_command(
        OUTPUT ${CPPREFERENCE_DOXYGEN_WEB_TAG}
        WORKING_DIRECTORY ${DOXYGEN_OUTPUT_DIR}
        VERBATIM
        COMMAND wget https://www-users.york.ac.uk/~od641/cppreference-doxygen-web.tag.xml
)

add_custom_command(
        OUTPUT ${DOXYGEN_SENTINEL}
        MAIN_DEPENDENCY Doxyfile
        DEPENDS ${DOXYGEN_SOURCES} ${CPPREFERENCE_DOXYGEN_WEB_TAG}
        VERBATIM
        WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
        COMMAND doxygen
        COMMAND touch ${DOXYGEN_SENTINEL}
)

add_custom_target(OptifolHTMLDocumentation DEPENDS
        ${CPPREFERENCE_DOXYGEN_WEB_TAG}
        ${DOXYGEN_SENTINEL}
)
```

3 CMakePresets.json

```
{
  "version": 3,
  "configurePresets": [
    {
      "name": "x64-linux-dynamic-vcpkg",
      "generator": "Ninja",
      "binaryDir": "${sourceDir}/build",
      "cacheVariables": {
        "VCPKG_TARGET_TRIPLET": "x64-linux-dynamic"
      },
      "toolchainFile": "$env{VCPKG_ROOT}/scripts/buildsystems/vcpkg.cmake",
      "warnings": {
        "unusedCli": false
      }
    }
  ]
}
```

4 UI

4.1 Application.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<interface>
    <menu id="top_menu">
        <submenu>
            <attribute name="label">File</attribute>
            <section>
                <submenu>
                    <attribute name="label">New</attribute>
                    <section>
                        <item>
                            <attribute name="label">Repository</attribute>
                            <attribute name="action">app.new_repository</attribute>
                        </item>
                    </section>
                    <section>
                        <item>
                            <attribute name="label">Project</attribute>
                            <attribute name="action">win.new_project</attribute>
                        </item>
                        <item>
                            <attribute name="label">Subsystem</attribute>
```

```
                        <attribute name="action">win.new_subsystem</attribute>
                    </item>
                    <item>
                        <attribute name="label">Requirement</attribute>
                        <attribute name="action">win.new_requirement</attribute>
                    </item>
                </section>
            </submenu>
            <item>
                <attribute name="label">Save Repository</attribute>
                <attribute name="action">app.save_repository</attribute>
            </item>
            <item>
                <attribute name="label">Open Repository</attribute>
                <attribute name="action">app.open_repository</attribute>
            </item>
            <submenu>
                <attribute name="label">Recent Repositories</attribute>
            </submenu>
        </section>
    </submenu>
    <submenu>
        <attribute name="label">Edit</attribute>
        <section>
            <submenu>
                <attribute name="label">Modify</attribute>
                <section>
                    <item>
                        <attribute name="label">Repository</attribute>
                        <attribute name="action">app.modify_repository</attribute>
                    </item>
                </section>
                <section>
                    <item>
                        <attribute name="label">Project</attribute>
                        <attribute name="action">win.modify_project</attribute>
                    </item>
                    <item>
                        <attribute name="label">Subsystem</attribute>
                        <attribute name="action">win.modify_subsystem</attribute>
                    </item>
                    <item>
                        <attribute name="label">Requirement</attribute>
                        <attribute name="action">win.modify_requirement</attribute>
                    </item>
                </section>
            </submenu>
        </section>
    </submenu>
    <submenu>
        <attribute name="label">Help</attribute>
        <item>
            <attribute name="label">Help</attribute>
            <attribute name="action">app.help</attribute>
        </item>
        <item>
            <attribute name="label">About</attribute>
            <attribute name="action">app.about</attribute>
        </item>
    </submenu>
</menu>

<object class="GtkAboutDialog" id="about_dialog">
    <property name="title">About Optifol</property>
    <property name="program-name">Optifol</property>
    <property name="version">In-Development Version</property>
    <property name="comments">A formal requirements-engineering framework</property>
    <property name="logo">resource:///uk/ac/york/www_users/od641/optifol/Images/wide_logo.png</property>

    <property name="website-label">Optifol Software Source Repository</property>
    <property name="website">https://github.com/oliverdixon/Optifol-Software</property>
    <property name="copyright">Copyright (C) 2025 Oliver Dixon. All rights reserved.</property>

    <property name="authors">Oliver Dixon &lt;od641@york.ac.uk&gt;</property>
    <property name="artists">Maia Dixon&#10;Theos Studio (SVG Repo Logo)</property>
</object>
</interface>
```

4.2 MainWindow

4.2.1 MainWindow.ui.in

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<interface>
    <menu id="structure_context_menu">
        <section>
            <item>
                <attribute name="label">New Project</attribute>
                <attribute name="action">win.new_project</attribute>
            </item>
            <item>
                <attribute name="label">New Subsystem</attribute>
                <attribute name="action">win.new_subsystem</attribute>
            </item>
            <item>
                <attribute name="label">Edit Details</attribute>
                <attribute name="action">win.edit_structure</attribute>
            </item>
            <item>
                <attribute name="label">Delete Structure</attribute>
                <attribute name="action">win.delete_structure</attribute>
            </item>
        </section>
    </menu>

    <menu id="requirement_context_menu">
        <section>
            <item>
                <attribute name="label">New Requirement</attribute>
                <attribute name="action">win.new_requirement</attribute>
            </item>
            <item>
                <attribute name="label">Edit Requirement</attribute>
                <attribute name="action">win.edit_requirement</attribute>
            </item>
            <item>
                <attribute name="label">Delete Requirement</attribute>
                <attribute name="action">win.delete_requirement</attribute>
            </item>
            <item>
                <attribute name="label">Duplicate Requirement</attribute>
                <attribute name="action">win.duplicate_requirement</attribute>
            </item>
        </section>
    </menu>

    <menu id="analysis_groups_context_menu">
        <section>
            <item>
                <attribute name="label">New Analysis Group</attribute>
                <attribute name="action">win.new_analysis_group</attribute>
            </item>
            <item>
                <attribute name="label">Edit Analysis Group</attribute>
                <attribute name="action">win.edit_analysis_group</attribute>
            </item>
            <item>
                <attribute name="label">Delete Analysis Group</attribute>
                <attribute name="action">win.delete_analysis_group</attribute>
            </item>
        </section>
    </menu>

    <menu id="test_groups_context_menu">
        <section>
            <item>
                <attribute name="label">New Test Group</attribute>
                <attribute name="action">win.new_test_group</attribute>
            </item>
            <item>
                <attribute name="label">Rename Test Group</attribute>
                <attribute name="action">win.rename_test_group</attribute>
            </item>
```

```
        <item>
            <attribute name="label">Delete Test Group</attribute>
            <attribute name="action">win.delete_test_group</attribute>
        </item>
        <item>
            <attribute name="label">Copy Requirement to Test Group</attribute>
            <attribute name="action">win.copy_to_test_group</attribute>
        </item>
        <item>
            <attribute name="label">Move Requirement to Test Group</attribute>
            <attribute name="action">win.move_to_test_group</attribute>
        </item>
        <item>
            <attribute name="label">Run Tests for Selected Test Group</attribute>
            <attribute name="action">win.run_tests</attribute>
        </item>
    </section>
</menu>

<menu id="reports_context_menu">
    <section>
        <item>
            <attribute name="label">Generate LaTeX/PDF</attribute>
            <attribute name="action">win.reports_generate_latex</attribute>
        </item>
        <item>
            <attribute name="label">Generate HTML</attribute>
            <attribute name="action">win.reports_generate_html</attribute>
        </item>
    </section>
</menu>

<!-- Project pane popovers -->
${NEW_PROJECT_POPOVER}
${NEW_SUBSYSTEM_POPOVER}
${EDIT_STRUCTURE_POPOVER}
${DELETE_STRUCTURE_POPOVER}

<!-- Requirements index area popovers -->
${NEW_REQUIREMENT_POPOVER}
${EDIT_REQUIREMENT_POPOVER}
${DUPLICATE_REQUIREMENT_POPOVER}
${DELETE_REQUIREMENT_POPOVER}
${MANAGE_TESTS_POPOVER}

<!-- Analysis area popovers -->
${NEW_ANALYSIS_GROUP_POPOVER}
${EDIT_ANALYSIS_GROUP_POPOVER}
${DELETE_ANALYSIS_GROUP_POPOVER}

<!-- Testing and compliance area popovers -->
${NEW_TEST_GROUP_POPOVER}
${RENAME_TEST_GROUP_POPOVER}
${DELETE_TEST_GROUP_POPOVER}
${COPY_TO_TEST_GROUP_POPOVER}
${MOVE_TO_TEST_GROUP_POPOVER}
${RUN_TESTS_POPOVER}

<!-- Releases and reports popovers -->
${GENERATE_LATEX_POPOVER}

<object class="GtkBox" id="root_grid">
    <property name="orientation">vertical</property>
    <child>
        <object class="GtkPaned">
            <property name="orientation">horizontal</property>

            <!--
              - The leftmost container, typically containing project-related views such as the interactive
              - hierarchical explorer view and any project metadata.
              -->
            <child>
                ${MAINWINDOW_PROJECT_PANE}
            </child>

            <!--
              - The central stack container, typically containing editor-related views such as the tabular
              - requirements indexer, project reports: anything extensively graphical or of premier
              - interest to the user.
              -->
            <child>
```

```
                    <object class="GtkBox">
                        <property name="hexpand">true</property>
                        <property name="orientation">vertical</property>
                        <child>
                            <object class="GtkStackSwitcher">
                                <property name="stack">central_stack</property>
                            </object>
                        </child>
                        <child>
                            <object class="GtkStack" id="central_stack">
                                <child>
                                    <object class="GtkStackPage">
                                        <property name="title">Requirements Index</property>
                                        <property name="child">
                                            ${REQUIREMENTS_INDEX_VIEW}
                                        </property>
                                    </object>
                                </child>
                                <child>
                                    <object class="GtkStackPage">
                                        <property name="title">Analysis and Optimisation</property>
                                        <property name="child">
                                            ${ANALYSIS_VIEW}
                                        </property>
                                    </object>
                                </child>
                                <child>
                                    <object class="GtkStackPage">
                                        <property name="title">Testing and Compliance</property>
                                        <property name="child">
                                            ${TESTING_VIEW}
                                        </property>
                                    </object>
                                </child>
                                <child>
                                    <object class="GtkStackPage">
                                        <property name="title">Releases and Reports</property>
                                        <property name="child">
                                            ${REPORTS_VIEW}
                                        </property>
                                    </object>
                                </child>
                            </object>
                        </child>
                    </object>
                </child>
            </object>
        </child>
        <child>
            <object class="GtkBox" id="system_status_bar">
                <style>
                    <class name="optifol_status_bar" />
                    <class name="optifol_boxed" />
                </style>
                <child>
                    <object class="GtkLabel">
                        <property name="label">Ready.</property>
                    </object>
                </child>
            </object>
        </child>
    </object>

    <object class="GtkFileDialog" id="generate_latex_output_path_chooser_dialog">
        <property name="accept-label" translatable="yes">Select Directory</property>
        <property name="modal">true</property>
        <property name="title">Select LaTeX/PDF Output Directory</property>
    </object>
</interface>
```

4.2.2 replacements.txt

```
MAINWINDOW_PROJECT_PANE=./ProjectPane/ProjectPane.ui
NEW_PROJECT_POPOVER=./ProjectPane/NewProjectPopover.ui
NEW_SUBSYSTEM_POPOVER=./ProjectPane/NewSubsystemPopover.ui
EDIT_STRUCTURE_POPOVER=./ProjectPane/EditStructurePopover.ui
DELETE_STRUCTURE_POPOVER=./ProjectPane/DeleteStructurePopover.ui

REQUIREMENTS_INDEX_VIEW=./RequirementsIndexArea/RequirementsIndexView.ui
```

```
NEW_REQUIREMENT_POPOVER=./RequirementsIndexArea/NewRequirementPopover.ui
DELETE_REQUIREMENT_POPOVER=./RequirementsIndexArea/DeleteRequirementPopover.ui
EDIT_REQUIREMENT_POPOVER=./RequirementsIndexArea/EditRequirementPopover.ui
DUPLICATE_REQUIREMENT_POPOVER=./RequirementsIndexArea/DuplicateRequirementPopover.ui
MANAGE_TESTS_POPOVER=./RequirementsIndexArea/ManageTestsPopover.ui

ANALYSIS_VIEW=./AnalysisArea/AnalysisView.ui
NEW_ANALYSIS_GROUP_POPOVER=./AnalysisArea/NewAnalysisGroupPopover.ui
EDIT_ANALYSIS_GROUP_POPOVER=./AnalysisArea/EditAnalysisGroupPopover.ui
DELETE_ANALYSIS_GROUP_POPOVER=./AnalysisArea/DeleteAnalysisGroupPopover.ui

TESTING_VIEW=./TestingArea/TestingView.ui
NEW_TEST_GROUP_POPOVER=./TestingArea/NewTestGroupPopover.ui
RENAME_TEST_GROUP_POPOVER=./TestingArea/RenameTestGroupPopover.ui
DELETE_TEST_GROUP_POPOVER=./TestingArea/DeleteTestGroupPopover.ui
COPY_TO_TEST_GROUP_POPOVER=./TestingArea/CopyToTestGroupPopover.ui
MOVE_TO_TEST_GROUP_POPOVER=./TestingArea/MoveToTestGroupPopover.ui
RUN_TESTS_POPOVER=./TestingArea/RunTests.ui

REPORTS_VIEW=./ReportsArea/ReportsArea.ui
GENERATE_LATEX_POPOVER=./ReportsArea/GenerateLaTeXPopover.ui
```

4.2.3 styles.css

```
/*
 * Copyright (c) All Rights Reserved
 * 2025 Oliver Dixon <od641@york.ac.uk>
 */

.optifol_status_bar {
    font-style: italic;
}

.optifol_boxed {
    padding: 0.5ex;
    border: 1px solid lightgray;
    border-radius: 5px;
}

.optifol_monospace {
    font-family: monospace;
}

.optifol_text_view {
    padding: 1.3ex;
}

.optifol_unknown {
    font-style: italic;
    color: gray;
}

.optifol_emphasised {
    font-style: italic;
}

.optifol_advisory {
    padding: 1ex;
    font-size: large;
    color: gray;
    margin: 1ex;
}

.optifol_grid_dialog > * {
    margin: 1ex;
}

.optifol_combo_box {
    padding: 0;
}
```

4.2.4 AnalysisArea

4.2.4.1 AnalysisView.ui

```
<!--
  - Copyright (c) All Rights Reserved
```

```
  – 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkBox">
    <child>
        <object class="GtkLabel" id="analysis_advice_unselected">
            <style>
                <class name="optifol_advisory"/>
                <class name="optifol_boxed"/>
            </style>
            <property name="use-markup">TRUE</property>
            <property name="justify">GTK_JUSTIFY_CENTER</property>
            <property name="halign">GTK_ALIGN_CENTER</property>
            <property name="valign">GTK_ALIGN_CENTER</property>
            <property name="hexpand">TRUE</property>
            <property name="vexpand">TRUE</property>
            <property name="max-width-chars">80</property>
            <property name="wrap">TRUE</property>
            <property name="label">
                <![CDATA[<b>The Analysis View is not applicable to Project-level scopes.</b>

The <i>Analysis View</i> allows organisation of Requirements defined in the selected Subsystem into
<i>Analysis Groups</i>.

Analysis Groups can be individually checked for logical consistency and scanned for redundant entries.

To prepare Analysis Groups, select a Subsystem from the left-hand <i>Project Explorer</i> pane and define some
Requirements in the <i>Requirements Index</i> view.]]>
            </property>
        </object>
    </child>
    <child>
        <object class="GtkPaned" id="analysis_index_content">
            <property name="visible">FALSE</property>
            <property name="orientation">vertical</property>
            <child>
                <object class="GtkBox">
                    <property name="orientation">vertical</property>
                    <child>
                        <object class="GtkBox">
                            <property name="halign">GTK_ALIGN_FILL</property>
                            <style>
                                <class name="toolbar"/>
                                <class name="optifol_boxed"/>
                            </style>
                            <child>
                                <object class="GtkMenuButton" id="new_analysis_group">
                                    <property name="icon-name">folder-new-symbolic</property>
                                    <property name="tooltip-text">New Analysis Group</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkMenuButton" id="edit_analysis_group">
                                    <property name="icon-name">document-edit-symbolic</property>
                                    <property name="tooltip-text">Edit Selected Analysis Group</property>
                                    <property name="sensitive">FALSE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkMenuButton" id="delete_analysis_group">
                                    <property name="icon-name">edit-delete-symbolic</property>
                                    <property name="tooltip-text">Delete Selected Analysis Group</property>
                                    <property name="sensitive">FALSE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkSeparator"/>
                            </child>
                            <child>
                                <object class="GtkMenuButton">
                                    <property name="icon-name">edit-copy-symbolic</property>
                                    <property name="tooltip-text">Copy Selected Requirement to Analysis
                                        Group
                                    </property>
                                    <property name="sensitive">FALSE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkMenuButton">
                                    <property name="icon-name">go-bottom-symbolic</property>
                                    <property name="tooltip-text">Move Selected Requirement to Analysis
```

```
                        Group
                    </property>
                    <property name="sensitive">FALSE</property>
                </object>
            </child>
            <child>
                <object class="GtkMenuButton">
                    <property name="icon-name">list-remove-symbolic</property>
                    <property name="tooltip-text">Remove Selected Requirement from Analysis
                        Group
                    </property>
                    <property name="sensitive">FALSE</property>
                </object>
            </child>
            <child>
                <object class="GtkSeparator"/>
            </child>
            <child>
                <object class="GtkMenuButton" id="create_new_query">
                    <property name="icon-name">list-add-symbolic</property>
                    <property name="tooltip-text">Create New Query</property>
                </object>
            </child>
            <child>
                <object class="GtkMenuButton" id="delete_selected_query">
                    <property name="icon-name">list-remove-symbolic</property>
                    <property name="tooltip-text">Remove Selected Query</property>
                    <property name="sensitive">FALSE</property>
                </object>
            </child>
            <child>
                <object class="GtkBox">
                    <property name="hexpand">TRUE</property>
                    <property name="halign">GTK_ALIGN_END</property>
                    <child>
                        <object class="GtkEntry">
                            <property name="sensitive">FALSE</property>
                            <property name="placeholder-text">Search by Name</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton">
                            <property name="sensitive">FALSE</property>
                            <property name="icon-name">edit-find-symbolic</property>
                            <property name="tooltip-text">Search Requirements</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton">
                            <property name="sensitive">FALSE</property>
                            <property name="icon-name">edit-clear-all-symbolic</property>
                            <property name="tooltip-text">Clear Search</property>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </child>
    <child>
        <object class="GtkScrolledWindow">
            <property name="vexpand">TRUE</property>
            <child>
                <object class="GtkColumnView" id="analysis_groups_view">
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">analysis_requirement_name</property>
                            <property name="title">Requirement</property>
                            <property name="resizable">TRUE</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">analysis_requirement_statement</property>
                            <property name="title">Statement</property>
                            <property name="resizable">TRUE</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">analysis_requirement_cnf</property>
                            <property name="title">Normalised Statement</property>
```

```
                                        <property name="resizable">TRUE</property>
                                    </object>
                                </child>
                            </object>
                        </child>
                    </object>
                </child>
            </object>
        </child>
    </object>
```

4.2.4.2 DeleteAnalysisGroupPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="delete_analysis_group_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Analysis Group</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="delete_analysis_group_property_name">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                </object>
            </child>
            <child>
                <object class="GtkSeparator" />
            </child>

            <child>
                <object class="GtkBox">
                    <property name="hexpand">TRUE</property>
                    <child>
                        <object class="GtkLabel">
                            <style>
                                <class name="optifol_emphasised" />
                            </style>
                            <property name="label">Are you sure?</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkBox">
                            <property name="hexpand">TRUE</property>
                            <property name="halign">GTK_ALIGN_END</property>
```

```
                        <style>
                            <class name="optifol_grid_dialog" />
                        </style>
                        <child>
                            <object class="GtkButton" id="delete_analysis_group_cancel">
                                <property name="icon-name">window-close-symbolic</property>
                                <property name="tooltip-text">Cancel Analysis Group Deletion</property>
                            </object>
                        </child>
                        <child>
                            <object class="GtkButton" id="delete_analysis_group_confirm">
                                <property name="icon-name">emblem-ok-symbolic</property>
                                <property name="tooltip-text">Confirm Analysis Group Deletion</property>
                            </object>
                        </child>
                    </object>
                </child>
            </object>
        </child>
    </object>
</property>
</object>
```

4.2.4.3 EditAnalysisGroupPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="edit_analysis_group_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Old Analysis Group Path</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="edit_analysis_group_property_old_path">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">New Analysis Group Path</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="edit_analysis_group_property_new_path">
                            <layout>
                                <property name="column">1</property>
                                <property name="row">1</property>
                            </layout>
```

```
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="edit_analysis_group_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Analysis Group Changes</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="edit_analysis_group_confirm">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Analysis Group Changes</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
  </property>
</object>
```

4.2.4.4 NewAnalysisGroupPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="new_analysis_group_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Analysis Group Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="new_analysis_group_property_name">
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                </object>
            </child>
            <child>
                <object class="GtkSeparator" />
            </child>
            <child>
                <object class="GtkBox">
                    <property name="halign">GTK_ALIGN_END</property>
```

```
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>
                    <child>
                        <object class="GtkButton" id="new_analysis_group_cancel">
                            <property name="icon-name">window-close-symbolic</property>
                            <property name="tooltip-text">Cancel Analysis Group Changes</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton" id="new_analysis_group_confirm">
                            <property name="icon-name">emblem-ok-symbolic</property>
                            <property name="tooltip-text">Confirm Analysis Group Changes</property>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </property>
</object>
```

4.2.5 ProjectPane

4.2.5.1 DeleteStructurePopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="delete_structure_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Structure Path</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="delete_structure_property_path">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                </object>
            </child>
            <child>
                <object class="GtkSeparator" />
            </child>

            <child>
                <object class="GtkBox">
                    <property name="hexpand">TRUE</property>
                    <child>
                        <object class="GtkLabel">
                            <style>
                                <class name="optifol_emphasised" />
                            </style>
                            <property name="label">Are you sure?</property>
                        </object>
```

```
                    </child>
                    <child>
                        <object class="GtkBox">
                            <property name="hexpand">TRUE</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <style>
                                <class name="optifol_grid_dialog" />
                            </style>
                            <child>
                                <object class="GtkButton" id="delete_structure_cancel">
                                    <property name="icon-name">window-close-symbolic</property>
                                    <property name="tooltip-text">Cancel Structure Deletion</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkButton" id="delete_structure_confirm">
                                    <property name="icon-name">emblem-ok-symbolic</property>
                                    <property name="tooltip-text">Confirm Structure Deletion</property>
                                </object>
                            </child>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </property>
</object>
```

4.2.5.2 EditStructurePopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="edit_structure_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Old Structure Path</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="edit_structure_property_old_path">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">New Structure Path</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>
                    <child>
```

```
                    <object class="GtkEntry" id="edit_structure_property_new_path">
                        <layout>
                            <property name="column">1</property>
                            <property name="row">1</property>
                        </layout>
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="edit_structure_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Structure Changes</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="edit_structure_confirm">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Structure Changes</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
  </property>
</object>
```

4.2.5.3 NewProjectPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="new_project_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Project Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="new_project_property_name">
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                </object>
            </child>
            <child>
```

```
            <object class="GtkSeparator" />
          </child>
          <child>
            <object class="GtkBox">
              <property name="halign">GTK_ALIGN_END</property>
              <style>
                <class name="optifol_grid_dialog" />
              </style>
              <child>
                <object class="GtkButton" id="new_project_cancel">
                  <property name="icon-name">window-close-symbolic</property>
                  <property name="tooltip-text">Cancel Project Changes</property>
                </object>
              </child>
              <child>
                <object class="GtkButton" id="new_project_confirm">
                  <property name="icon-name">emblem-ok-symbolic</property>
                  <property name="tooltip-text">Confirm Project Details</property>
                </object>
              </child>
            </object>
          </child>
        </object>
    </property>
</object>
```

4.2.5.4 NewSubsystemPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="new_subsystem_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Path</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="new_subsystem_property_path">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Subsystem Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="new_subsystem_property_name">
                            <layout>
```

```
                            <property name="column">1</property>
                            <property name="row">1</property>
                        </layout>
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="new_subsystem_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Subsystem Changes</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="new_subsystem_confirm">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Subsystem Details</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
  </property>
</object>
```

4.2.5.5 ProjectPane.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkBox">
    <property name="orientation">vertical</property>
    <child>
        <object class="GtkDropDown" id="project_pane_switcher">
            <property name="model">
                <object class="GtkStringList">
                    <items>
                        <item>Project Explorer</item>
                        <item>System Metadata</item>
                    </items>
                </object>
            </property>
        </object>
    </child>
    <child>
        <object class="GtkStack" id="project_pane_stack">
            <child>
                <object class="GtkStackPage">
                    <property name="name">project_explorer</property>
                    <property name="child">
                        <object class="GtkBox">
                            <property name="orientation">vertical</property>
                            <child>
                                <object class="GtkBox">
                                    <property name="halign">GTK_ALIGN_FILL</property>
                                    <style>
                                        <class name="toolbar"/>
                                        <class name="optifol_boxed"/>
                                    </style>
                                    <child>
                                        <object class="GtkMenuButton" id="new_project">
                                            <property name="icon-name">document-new-symbolic</property>
                                            <property name="tooltip-text">New Project</property>
                                        </object>
                                    </child>
                                    <child>
                                        <object class="GtkMenuButton" id="new_subsystem">
```

```
                                    <property name="icon-name">window-new-symbolic</property>
                                    <property name="tooltip-text">New Subsystem</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkMenuButton" id="edit_structure">
                                    <property name="icon-name">
                                        accessories-text-editor-symbolic</property>
                                    <property name="tooltip-text">Edit Structure</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkMenuButton" id="delete_structure">
                                    <property name="icon-name">edit-delete-symbolic</property>
                                    <property name="tooltip-text">
                                        Delete Structure from Here</property>
                                </object>
                            </child>
                        </object>
                    </child>
                    <child>
                        <object class="GtkScrolledWindow">
                            <property name="vexpand">TRUE</property>
                            <child>
                                <object class="GtkListView" id="project_view">
                                    <property name="single-click-activate">FALSE</property>
                                </object>
                            </child>
                        </object>
                    </child>
                </object>
            </property>
        </object>
    </child>
    <child>
        <object class="GtkStackPage">
            <property name="name">project_metadata</property>
            <property name="child">
                <object class="GtkScrolledWindow">
                    <property name="vexpand">TRUE</property>
                    <child>
                        <object class="GtkColumnView">
                            <style>
                                <class name="data-table"/>
                            </style>
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">project_metadata_attribute</property>
                                    <property name="title">Attribute</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">project_metadata_value</property>
                                    <property name="title">Value</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                        </object>
                    </child>
                </object>
            </property>
        </object>
    </child>
</object>
```

4.2.6 ReportsArea

4.2.6.1 GenerateLaTeXPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="reports_generate_latex_popover">
```

```
<property name="child">
    <object class="GtkBox">
        <style>
            <class name="optifol_grid_dialog" />
        </style>
        <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
        <child>
            <object class="GtkGrid">
                <style>
                    <class name="optifol_grid_dialog" />
                </style>

                <child>
                    <object class="GtkLabel">
                        <property name="label">Output Directory</property>
                        <property name="halign">GTK_ALIGN_END</property>
                        <layout>
                            <property name="column">0</property>
                            <property name="row">0</property>
                        </layout>
                    </object>
                </child>
                <child>
                    <object class="GtkEntry" id="generate_latex_output_path">
                        <layout>
                            <property name="column">1</property>
                            <property name="row">0</property>
                        </layout>
                        <property name="editable">FALSE</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="generate_latex_output_path_chooser_button">
                        <layout>
                            <property name="column">2</property>
                            <property name="row">0</property>
                        </layout>
                        <property name="icon-name">folder-open-symbolic</property>
                        <property name="tooltip-text">Select LaTeX/PDF Output Directory</property>
                    </object>
                </child>
            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox" id="generate_latex_details_container">
                <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
                <child>
                    <object class="GtkScrolledWindow">
                        <property name="vexpand">false</property>
                        <property name="hexpand">true</property>
                        <property name="propagate-natural-height">true</property>
                        <property name="propagate-natural-width">true</property>
                        <child>
                            <object class="GtkTextView" id="generate_latex_output">
                                <style>
                                    <class name="optifol_monospace" />
                                </style>
                                <property name="editable">FALSE</property>
                            </object>
                        </child>
                    </object>
                </child>
                <child>
                    <object class="GtkSeparator" />
                </child>
            </object>
        </child>
        <child>
            <object class="GtkBox">
                <property name="hexpand">TRUE</property>
                <child>
                    <object class="GtkCheckButton" id="generate_latex_show_details">
                        <property name="label">Show compiler output</property>
                        <property name="active">TRUE</property>
                    </object>
                </child>
                <child>
                    <object class="GtkBox">
```

```
                        <property name="hexpand">TRUE</property>
                        <property name="halign">GTK_ALIGN_END</property>
                        <style>
                            <class name="optifol_grid_dialog" />
                        </style>
                        <child>
                            <object class="GtkButton" id="generate_latex_cancel">
                                <property name="icon-name">window-close-symbolic</property>
                                <property name="tooltip-text">Cancel LaTeX Generation</property>
                            </object>
                        </child>
                        <child>
                            <object class="GtkButton" id="generate_latex_confirm">
                                <property name="icon-name">emblem-ok-symbolic</property>
                                <property name="tooltip-text">Generate LaTeX and PDF</property>
                            </object>
                        </child>
                    </object>
                </child>
            </object>
        </child>
    </object>
</property>
</object>
```

4.2.6.2 ReportsArea.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkBox">
    <child>
        <object class="GtkLabel" id="reports_advice_unselected">
            <style>
                <class name="optifol_advisory"/>
                <class name="optifol_boxed"/>
            </style>
            <property name="use-markup">TRUE</property>
            <property name="justify">GTK_JUSTIFY_CENTER</property>
            <property name="halign">GTK_ALIGN_CENTER</property>
            <property name="valign">GTK_ALIGN_CENTER</property>
            <property name="hexpand">TRUE</property>
            <property name="vexpand">TRUE</property>
            <property name="max-width-chars">80</property>
            <property name="wrap">TRUE</property>
            <property name="label">
                <![CDATA[<b>The Releases and Reports View is not applicable to Project-level scopes.</b>

The <i>Releases and Reports View</i> allows automatic generation of HTML and LaTeX/PDF reports based on the
Requirements and related Analysis of a Subsystem.

The content of exported Reports can be cherry-picked, and formatting can be extensively customised.

To prepare a Report, select a Subsystem from the left-hand <i>Project Explorer</i> pane and define some
Requirements in the <i>Requirements Index</i> view.]]>
            </property>
        </object>
    </child>
    <child>
        <object class="GtkBox" id="reports_content">
            <property name="visible">FALSE</property>
            <property name="orientation">vertical</property>
            <child>
                <object class="GtkBox">
                    <property name="halign">GTK_ALIGN_FILL</property>
                    <style>
                        <class name="toolbar"/>
                        <class name="optifol_boxed"/>
                    </style>
                    <child>
                        <object class="GtkMenuButton" id="reports_modify_metadata">
                            <property name="icon-name">document-edit-symbolic</property>
                            <property name="tooltip-text">Edit Report Metadata</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkSeparator" />
```

```
                    </child>
                    <child>
                        <object class="GtkMenuButton" id="reports_include_element">
                            <property name="icon-name">tab-new-symbolic</property>
                            <property name="tooltip-text">Include Selected Element</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkMenuButton" id="reports_exclude_element">
                            <property name="icon-name">list-remove-symbolic</property>
                            <property name="tooltip-text">Exclude Selected Element</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkSeparator" />
                    </child>
                    <child>
                        <object class="GtkMenuButton" id="reports_generate_latex">
                            <property name="icon-name">emblem-documents-symbolic</property>
                            <property name="tooltip-text">Generate LaTeX/PDF Report</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkMenuButton" id="reports_generate_html">
                            <property name="icon-name">emblem-documents-symbolic</property>
                            <property name="tooltip-text">Generate HTML Report</property>
                        </object>
                    </child>
                </object>
            </child>
            <child>
                <object class="GtkScrolledWindow">
                    <property name="vexpand">TRUE</property>
                    <child>
                        <object class="GtkColumnView" id="reports_elements_view">
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">reports_element_name</property>
                                    <property name="title">Element</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">reports_item_count</property>
                                    <property name="title">Item Count</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </child>
</object>
```

4.2.7 RequirementsIndexArea

4.2.7.1 DeleteRequirementPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="delete_requirement_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>
```

```
                    <child>
                        <object class="GtkLabel">
                            <property name="label">Requirement Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="delete_requirement_property_name">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                </object>
            </child>
            <child>
                <object class="GtkSeparator" />
            </child>

            <child>
                <object class="GtkBox">
                    <property name="hexpand">TRUE</property>
                    <child>
                        <object class="GtkLabel">
                            <style>
                                <class name="optifol_emphasised" />
                            </style>
                            <property name="label">Are you sure?</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkBox">
                            <property name="hexpand">TRUE</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <style>
                                <class name="optifol_grid_dialog" />
                            </style>
                            <child>
                                <object class="GtkButton" id="delete_requirement_cancel">
                                    <property name="icon-name">window-close-symbolic</property>
                                    <property name="tooltip-text">Cancel Requirement Deletion</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkButton" id="delete_requirement_confirm">
                                    <property name="icon-name">emblem-ok-symbolic</property>
                                    <property name="tooltip-text">Confirm Requirement Deletion</property>
                                </object>
                            </child>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </property>
</object>
```

4.2.7.2 DuplicateRequirementPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="duplicate_requirement_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
```

```
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Old Requirement Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="duplicate_requirement_property_old_name">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">New Requirement Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="duplicate_requirement_property_new_name">
                            <layout>
                                <property name="column">1</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>

                </object>
            </child>
            <child>
                <object class="GtkSeparator" />
            </child>
            <child>
                <object class="GtkBox">
                    <property name="halign">GTK_ALIGN_END</property>
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>
                    <child>
                        <object class="GtkButton" id="duplicate_requirement_cancel">
                            <property name="icon-name">window-close-symbolic</property>
                            <property name="tooltip-text">Cancel Requirement Duplication</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton" id="duplicate_requirement_confirm">
                            <property name="icon-name">emblem-ok-symbolic</property>
                            <property name="tooltip-text">Confirm Requirement Duplication</property>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </property>
</object>
```

4.2.7.3 EditRequirementPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->
```

```
<object class="GtkPopover" id="edit_requirement_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Requirement Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="edit_requirement_property_name">
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Description</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkTextView" id="edit_requirement_property_description">
                            <style>
                                <class name="optifol_boxed" />
                                <class name="optifol_text_view" />
                            </style>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">FOL Sentence</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">2</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="edit_requirement_property_sentence">
                            <style>
                                <class name="optifol_monospace" />
                            </style>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">2</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Associated Test</property>
```

```
                        <property name="halign">GTK_ALIGN_END</property>
                        <layout>
                            <property name="column">0</property>
                            <property name="row">3</property>
                        </layout>
                    </object>
                </child>
                <child>
                    <object class="GtkEntry" id="edit_requirement_property_test">
                        <property name="placeholder-text">None</property>
                        <layout>
                            <property name="column">1</property>
                            <property name="row">3</property>
                        </layout>
                    </object>
                </child>

                <child>
                    <object class="GtkLabel">
                        <property name="label">Priority</property>
                        <property name="halign">GTK_ALIGN_END</property>
                        <layout>
                            <property name="column">0</property>
                            <property name="row">4</property>
                        </layout>
                    </object>
                </child>
                <child>
                    <object class="GtkDropDown" id="edit_requirement_property_priority">
                        <style>
                            <class name="optifol_boxed" />
                            <class name="optifol_combo_box" />
                        </style>
                        <property name="model">
                            <object class="GtkStringList">
                                <items>
                                    <item>Optional</item>
                                    <item>Low</item>
                                    <item>Normal</item>
                                    <item>High</item>
                                    <item>Critical</item>
                                </items>
                            </object>
                        </property>
                        <layout>
                            <property name="column">1</property>
                            <property name="row">4</property>
                        </layout>
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="hexpand">TRUE</property>
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="edit_requirement_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Requirement Changes</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="edit_requirement_confirm">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Requirement Details</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
</property>
</object>
```

4.2.7.4 ManageTestsPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="manage_tests_popover">
    <property name="position">GTK_POS_RIGHT</property>
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">vertical</property>
            <child>
                <object class="GtkColumnView" id="manage_tests_view">
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">manage_tests_target_exe</property>
                            <property name="title">Target Executable</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">manage_tests_fixture</property>
                            <property name="title">Test Fixture</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">manage_tests_name</property>
                            <property name="title">Test Name</property>
                        </object>
                    </child>
                </object>
            </child>
            <child>
                <object class="GtkSeparator" />
            </child>
            <child>
                <object class="GtkBox">
                    <property name="hexpand">TRUE</property>
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>
                    <child>
                        <object class="GtkButton" id="manage_tests_new_test">
                            <property name="icon-name">list-add-symbolic</property>
                            <property name="tooltip-text">Add New Test</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton" id="manage_tests_duplicate_test">
                            <property name="icon-name">edit-copy-symbolic</property>
                            <property name="tooltip-text">Duplicate Selected Test</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton" id="manage_tests_delete_test">
                            <property name="icon-name">edit-delete-symbolic</property>
                            <property name="tooltip-text">Delete Selected Test</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkBox">
                            <property name="hexpand">TRUE</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <child>
                                <object class="GtkButton" id="manage_tests_confirm">
                                    <property name="icon-name">emblem-ok-symbolic</property>
                                    <property name="tooltip-text">Update Tests</property>
                                </object>
                            </child>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </property>
```

```
</object>
```

4.2.7.5 NewRequirementPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="new_requirement_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Requirement Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="new_requirement_property_name">
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Description</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkTextView" id="new_requirement_property_description">
                            <style>
                                <class name="optifol_boxed" />
                                <class name="optifol_text_view" />
                            </style>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">1</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">FOL Sentence</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">2</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="new_requirement_property_sentence">
                            <style>
                                <class name="optifol_monospace" />
                            </style>
```

```
                <layout>
                    <property name="column">1</property>
                    <property name="row">2</property>
                </layout>
            </object>
        </child>

        <child>
            <object class="GtkLabel">
                <property name="label">Software Tests</property>
                <property name="halign">GTK_ALIGN_END</property>
                <layout>
                    <property name="column">0</property>
                    <property name="row">3</property>
                </layout>
            </object>
        </child>
        <child>
            <object class="GtkBox">
                <property name="orientation">horizontal</property>
                <child>
                    <object class="GtkEntry" id="new_requirement_test_count">
                        <property name="placeholder-text">No tests defined</property>
                        <property name="sensitive">FALSE</property>
                    </object>
                </child>
                <child>
                    <object class="GtkMenuButton" id="new_requirement_manage_tests">
                        <property name="icon-name">document-edit-symbolic</property>
                        <property name="tooltip-text">Manage Associated Software Tests</property>
                    </object>
                </child>
                <layout>
                    <property name="column">1</property>
                    <property name="row">3</property>
                </layout>
            </object>
        </child>

        <child>
            <object class="GtkLabel">
                <property name="label">Priority</property>
                <property name="halign">GTK_ALIGN_END</property>
                <layout>
                    <property name="column">0</property>
                    <property name="row">4</property>
                </layout>
            </object>
        </child>
        <child>
            <object class="GtkDropDown" id="new_requirement_property_priority">
                <style>
                    <class name="optifol_boxed" />
                    <class name="optifol_combo_box" />
                </style>
                <property name="model">
                    <object class="GtkStringList">
                        <items>
                            <item>Optional</item>
                            <item>Low</item>
                            <item>Normal</item>
                            <item>High</item>
                            <item>Critical</item>
                        </items>
                    </object>
                </property>
                <layout>
                    <property name="column">1</property>
                    <property name="row">4</property>
                </layout>
            </object>
        </child>

    </object>
</child>
<child>
    <object class="GtkSeparator" />
</child>
<child>
    <object class="GtkBox">
        <property name="hexpand">TRUE</property>
```

```
                    <property name="halign">GTK_ALIGN_END</property>
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>
                    <child>
                        <object class="GtkButton" id="new_requirement_cancel">
                            <property name="icon-name">window-close-symbolic</property>
                            <property name="tooltip-text">Cancel Requirement Changes</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton" id="new_requirement_confirm">
                            <property name="icon-name">emblem-ok-symbolic</property>
                            <property name="sensitive">FALSE</property>
                            <property name="tooltip-text">Confirm Requirement Details</property>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </property>
</object>
```

4.2.7.6 RequirementsIndexView.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkBox">
    <property name="orientation">vertical</property>
    <child>
        <object class="GtkLabel" id="requirements_index_advice_unselected">
            <style>
                <class name="optifol_advisory"/>
                <class name="optifol_boxed"/>
            </style>
            <property name="use-markup">TRUE</property>
            <property name="justify">GTK_JUSTIFY_CENTER</property>
            <property name="halign">GTK_ALIGN_CENTER</property>
            <property name="valign">GTK_ALIGN_CENTER</property>
            <property name="hexpand">TRUE</property>
            <property name="vexpand">TRUE</property>
            <property name="max-width-chars">80</property>
            <property name="wrap">TRUE</property>
            <property name="label">
                <![CDATA[<b>The Requirements Index is not applicable to Project-level scopes.</b>

The <i>Requirements Index</i> presents a tabular view of all Requirements within the active Subsystem.

From the Index, you can review, modify, search, and filter Requirements in preparation for formal analysis, compliance verification, and report generation.

To view an Index, select a Subsystem from the left-hand <i>Project Explorer</i> pane.]]>
            </property>
        </object>
    </child>
    <child>
        <object class="GtkBox" id="requirements_index_content">
            <property name="visible">FALSE</property>
            <property name="orientation">vertical</property>
            <child>
                <object class="GtkBox">
                    <property name="halign">GTK_ALIGN_FILL</property>
                    <style>
                        <class name="toolbar"/>
                        <class name="optifol_boxed"/>
                    </style>
                    <child>
                        <object class="GtkMenuButton" id="new_requirement">
                            <property name="icon-name">tab-new-symbolic</property>
                            <property name="tooltip-text">New Requirement</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkMenuButton" id="edit_requirement">
                            <property name="icon-name">document-edit-symbolic</property>
                            <property name="tooltip-text">Edit Selected Requirement</property>
```

```
                    <property name="sensitive">FALSE</property>
                </object>
            </child>
            <child>
                <object class="GtkMenuButton" id="delete_requirement">
                    <property name="icon-name">edit-delete-symbolic</property>
                    <property name="tooltip-text">Delete Selected Requirements</property>
                    <property name="sensitive">FALSE</property>
                </object>
            </child>
            <child>
                <object class="GtkMenuButton" id="duplicate_requirement">
                    <property name="icon-name">edit-copy-symbolic</property>
                    <property name="tooltip-text">Duplicate Selected Requirement</property>
                    <property name="sensitive">FALSE</property>
                </object>
            </child>
            <child>
                <object class="GtkBox">
                    <property name="hexpand">TRUE</property>
                    <property name="halign">GTK_ALIGN_END</property>
                    <child>
                        <object class="GtkEntry">
                            <property name="sensitive">FALSE</property>
                            <property name="placeholder-text">Search by Name</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton">
                            <property name="sensitive">FALSE</property>
                            <property name="icon-name">edit-find-symbolic</property>
                            <property name="tooltip-text">Search Requirements</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton">
                            <property name="sensitive">FALSE</property>
                            <property name="icon-name">edit-clear-all-symbolic</property>
                            <property name="tooltip-text">Clear Search</property>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </child>
    <child>
        <object class="GtkScrolledWindow">
            <property name="vexpand">TRUE</property>
            <child>
                <object class="GtkColumnView" id="requirements_view">
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">requirement_name</property>
                            <property name="title">Name</property>
                            <property name="resizable">TRUE</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">requirement_description</property>
                            <property name="title">Description</property>
                            <property name="resizable">TRUE</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">requirement_statement</property>
                            <property name="title">Statement</property>
                            <property name="resizable">TRUE</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">requirement_priority</property>
                            <property name="title">Priority</property>
                            <property name="resizable">TRUE</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkColumnViewColumn">
                            <property name="id">requirement_test</property>
```

```
                                <property name="title">Test</property>
                                <property name="resizable">TRUE</property>
                            </object>
                        </child>
                        <child>
                            <object class="GtkColumnViewColumn">
                                <property name="id">requirement_stakeholder</property>
                                <property name="title">Stakeholder</property>
                                <property name="resizable">TRUE</property>
                            </object>
                        </child>
                        <child>
                            <object class="GtkColumnViewColumn">
                                <property name="id">requirement_created</property>
                                <property name="title">Created</property>
                                <property name="resizable">TRUE</property>
                            </object>
                        </child>
                        <child>
                            <object class="GtkColumnViewColumn">
                                <property name="id">requirement_modified</property>
                                <property name="title">Modified</property>
                                <property name="resizable">TRUE</property>
                            </object>
                        </child>
                    </object>
                </child>
            </object>
        </child>
    </object>
</child>
</object>
```

4.2.8 TestingArea

4.2.8.1 CopyToTestGroupPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="copy_to_test_group_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Requirement Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="copy_to_test_group_requirement">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">New Test Group</property>
                            <property name="halign">GTK_ALIGN_END</property>
```

```
                        <layout>
                            <property name="column">0</property>
                            <property name="row">1</property>
                        </layout>
                    </object>
                </child>
                <child>
                    <object class="GtkDropDown" id="copy_to_test_group_new_group">
                        <style>
                            <class name="optifol_boxed" />
                            <class name="optifol_combo_box" />
                        </style>
                        <layout>
                            <property name="column">1</property>
                            <property name="row">1</property>
                        </layout>
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="copy_to_test_group_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Requirement Copy</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="copy_to_test_group_confirm">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Requirement Copy</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
</property>
</object>
```

4.2.8.2 DeleteTestGroupPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="delete_test_group_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Test Group Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
```

```
                    <object class="GtkEntry" id="delete_test_group_name">
                        <property name="sensitive">FALSE</property>
                        <layout>
                            <property name="column">1</property>
                            <property name="row">0</property>
                        </layout>
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="delete_test_group_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Test Group Deletion</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="delete_test_group_confirm">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Test Group Deletion</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
  </property>
</object>
```

4.2.8.3 MoveToTestGroupPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="move_to_test_group_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Requirement Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="move_to_test_group_requirement">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>

                    <child>
```

```
                    <object class="GtkLabel">
                        <property name="label">New Test Group</property>
                        <property name="halign">GTK_ALIGN_END</property>
                        <layout>
                            <property name="column">0</property>
                            <property name="row">1</property>
                        </layout>
                    </object>
                </child>
                <child>
                    <object class="GtkDropDown" id="move_to_test_group_new_group">
                        <style>
                            <class name="optifol_boxed" />
                            <class name="optifol_combo_box" />
                        </style>
                        <layout>
                            <property name="column">1</property>
                            <property name="row">1</property>
                        </layout>
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="move_to_test_group_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Requirement Move</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="move_to_test_group_confirm">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Requirement Move</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
</property>
</object>
```

4.2.8.4 NewTestGroupPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="new_test_group_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Test Group Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
```

```
                    </object>
                </child>
                <child>
                    <object class="GtkEntry" id="new_test_group_name">
                        <layout>
                            <property name="column">1</property>
                            <property name="row">0</property>
                        </layout>
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="new_test_group_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Test Group Changes</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="new_test_group_confirm">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Test Group Changes</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
  </property>
</object>
```

4.2.8.5 RenameTestGroupPopover.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="rename_test_group_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>

                    <child>
                        <object class="GtkLabel">
                            <property name="label">Old Test Group Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
                    <child>
                        <object class="GtkEntry" id="rename_test_group_old_name">
                            <property name="sensitive">FALSE</property>
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
                    </child>
```

```
                <child>
                    <object class="GtkLabel">
                        <property name="label">New Test Group Name</property>
                        <property name="halign">GTK_ALIGN_END</property>
                        <layout>
                            <property name="column">0</property>
                            <property name="row">1</property>
                        </layout>
                    </object>
                </child>
                <child>
                    <object class="GtkEntry" id="rename_test_group_new_name">
                        <layout>
                            <property name="column">1</property>
                            <property name="row">1</property>
                        </layout>
                    </object>
                </child>

            </object>
        </child>
        <child>
            <object class="GtkSeparator" />
        </child>
        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_END</property>
                <style>
                    <class name="optifol_grid_dialog" />
                </style>
                <child>
                    <object class="GtkButton" id="rename_test_group_cancel">
                        <property name="icon-name">window-close-symbolic</property>
                        <property name="tooltip-text">Cancel Test Group Rename</property>
                    </object>
                </child>
                <child>
                    <object class="GtkButton" id="rename_test_group_confirm">
                        <property name="sensitive">FALSE</property>
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Confirm Test Group Rename</property>
                    </object>
                </child>
            </object>
        </child>
    </object>
</property>
</object>
```

4.2.8.6 RunTests.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkPopover" id="run_tests_popover">
    <property name="child">
        <object class="GtkBox">
            <style>
                <class name="optifol_grid_dialog" />
            </style>
            <property name="orientation">GTK_ORIENTATION_VERTICAL</property>

            <child>
                <object class="GtkGrid">
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>
                    <child>
                        <object class="GtkLabel">
                            <property name="label">Test Group Name</property>
                            <property name="halign">GTK_ALIGN_END</property>
                            <layout>
                                <property name="column">0</property>
                                <property name="row">0</property>
                            </layout>
                        </object>
```

```
                    </child>
                    <child>
                        <object class="GtkEntry" id="test_group_name">
                            <layout>
                                <property name="column">1</property>
                                <property name="row">0</property>
                            </layout>
                            <property name="sensitive">FALSE</property>
                        </object>
                    </child>
                </object>
            </child>

            <child>
                <object class="GtkSeparator" />
            </child>

            <child>
                <object class="GtkBox">
                    <property name="hexpand">TRUE</property>
                    <property name="halign">GTK_ALIGN_END</property>
                    <style>
                        <class name="optifol_grid_dialog" />
                    </style>
                    <child>
                        <object class="GtkButton" id="run_tests_cancel">
                            <property name="icon-name">window-close-symbolic</property>
                            <property name="tooltip-text">Cancel Test Execution</property>
                        </object>
                    </child>
                    <child>
                        <object class="GtkButton" id="run_tests_confirm">
                            <property name="icon-name">emblem-ok-symbolic</property>
                            <property name="tooltip-text">Run Tests</property>
                        </object>
                    </child>
                </object>
            </child>

        </object>
    </property>
</object>
```

4.2.8.7 TestingView.ui

```
<!--
  - Copyright (c) All Rights Reserved
  - 2025 Oliver Dixon <od641@york.ac.uk>
  -->

<object class="GtkBox">
    <child>
        <object class="GtkLabel" id="testing_advice_unselected">
            <style>
                <class name="optifol_advisory"/>
                <class name="optifol_boxed"/>
            </style>
            <property name="use-markup">TRUE</property>
            <property name="justify">GTK_JUSTIFY_CENTER</property>
            <property name="halign">GTK_ALIGN_CENTER</property>
            <property name="valign">GTK_ALIGN_CENTER</property>
            <property name="hexpand">TRUE</property>
            <property name="vexpand">TRUE</property>
            <property name="max-width-chars">80</property>
            <property name="wrap">TRUE</property>
            <property name="label">
                <![CDATA[<b>The Testing View is not applicable to Project-level scopes.</b>

The <i>Testing View</i> allows organisation of Test-associated Requirements defined in the selected Subsystem into <i>Test Groups</i>.

Test Groups can be individually subject to execution under an external unit testing framework, such as Google Test, to determine software-level satisfaction of Requirements.

To prepare Test Groups, select a Subsystem from the left-hand <i>Project Explorer</i> pane and define some Requirements in the <i>Requirements Index</i> view.]]>
            </property>
        </object>
    </child>
```

```
<child>
    <object class="GtkBox" id="testing_content">
        <property name="orientation">vertical</property>

        <child>
            <object class="GtkBox">
                <property name="halign">GTK_ALIGN_FILL</property>
                <style>
                    <class name="toolbar"/>
                    <class name="optifol_boxed"/>
                </style>
                <child>
                    <object class="GtkMenuButton" id="new_test_group">
                        <property name="icon-name">folder-new-symbolic</property>
                        <property name="tooltip-text">New Test Group</property>
                    </object>
                </child>
                <child>
                    <object class="GtkMenuButton" id="rename_test_group">
                        <property name="icon-name">document-edit-symbolic</property>
                        <property name="tooltip-text">Rename Test Group</property>
                        <property name="sensitive">FALSE</property>
                    </object>
                </child>
                <child>
                    <object class="GtkMenuButton" id="delete_test_group">
                        <property name="icon-name">edit-delete-symbolic</property>
                        <property name="tooltip-text">Delete Test Group</property>
                        <property name="sensitive">FALSE</property>
                    </object>
                </child>
                <child>
                    <object class="GtkSeparator"/>
                </child>
                <child>
                    <object class="GtkMenuButton" id="copy_to_test_group">
                        <property name="icon-name">edit-copy-symbolic</property>
                        <property name="tooltip-text">Copy Selected Requirement to Test Group</property>
                        <property name="sensitive">FALSE</property>
                    </object>
                </child>
                <child>
                    <object class="GtkMenuButton" id="move_to_test_group">
                        <property name="icon-name">go-bottom-symbolic</property>
                        <property name="tooltip-text">Move Selected Requirement to Test Group</property>
                        <property name="sensitive">FALSE</property>
                    </object>
                </child>
                <child>
                    <object class="GtkSeparator"/>
                </child>
                <child>
                    <object class="GtkMenuButton" id="run_tests">
                        <property name="icon-name">emblem-ok-symbolic</property>
                        <property name="tooltip-text">Re-Run Selected Test Group</property>
                    </object>
                </child>
                <child>
                    <object class="GtkBox">
                        <property name="hexpand">TRUE</property>
                        <property name="halign">GTK_ALIGN_END</property>
                        <child>
                            <object class="GtkEntry">
                                <property name="sensitive">FALSE</property>
                                <property name="placeholder-text">Search by Name</property>
                            </object>
                        </child>
                        <child>
                            <object class="GtkButton">
                                <property name="sensitive">FALSE</property>
                                <property name="icon-name">edit-find-symbolic</property>
                                <property name="tooltip-text">Search Requirements</property>
                            </object>
                        </child>
                        <child>
                            <object class="GtkButton">
                                <property name="sensitive">FALSE</property>
                                <property name="icon-name">edit-clear-all-symbolic</property>
                                <property name="tooltip-text">Clear Search</property>
                            </object>
                        </child>
```

```
                </object>
            </child>
        </object>
    </child>

    <child>
        <object class="GtkPaned">
            <property name="orientation">vertical</property>
            <property name="position">400</property>
            <child>
                <object class="GtkScrolledWindow">
                    <property name="vexpand">TRUE</property>
                    <child>
                        <object class="GtkColumnView" id="test_groups_view">
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">test_requirement_name</property>
                                    <property name="title">Test Case Name</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">test_target_executable</property>
                                    <property name="title">Target Executable</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">test_fixture</property>
                                    <property name="title">Test Fixture</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">test_status</property>
                                    <property name="title">Test Status</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                        </object>
                    </child>
                </object>
            </child>
            <child>
                <object class="GtkScrolledWindow" id="test_groups_failed_container">
                    <property name="vexpand">TRUE</property>
                    <property name="visible">FALSE</property>
                    <child>
                        <object class="GtkColumnView" id="test_groups_failed_view">
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">failed_test_message</property>
                                    <property name="title">Failed Test Message</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">failed_test_source_file</property>
                                    <property name="title">Source File</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                            <child>
                                <object class="GtkColumnViewColumn">
                                    <property name="id">failed_test_source_line</property>
                                    <property name="title">Source Line</property>
                                    <property name="resizable">TRUE</property>
                                </object>
                            </child>
                        </object>
                    </child>
                </object>
            </child>
        </object>
    </child>

</object>
```

```
    </child>
</object>
```

5 vcpkg.json

```
{
  "name": "optifol",
  "version-date": "2025-09-25",
  "dependencies": [
    {
      "name": "gtkmm",
      "version>=": "4.14.0"
    },
    {
      "name": "log4cxx",
      "version>=": "1.5.0"
    },
    {
      "name": "rapidjson",
      "version>=": "2025-02-26"
    }
  ]
}
```

6 .clang-format

```
---
Language: Cpp
BasedOnStyle: LLVM
AccessModifierOffset: -4
AlignConsecutiveAssignments: false
AlignConsecutiveDeclarations: false
AlignOperands: false
AlignTrailingComments: false
AlwaysBreakTemplateDeclarations: Yes
BraceWrapping:
  AfterCaseLabel: false
  AfterClass: true
  AfterControlStatement: false
  AfterEnum: true
  AfterFunction: true
  AfterNamespace: true
  AfterStruct: true
  AfterUnion: true
  AfterExternBlock: true
  BeforeCatch: false
  BeforeElse: false
  BeforeLambdaBody: true
  BeforeWhile: false
  SplitEmptyFunction: true
  SplitEmptyRecord: true
  SplitEmptyNamespace: true
BreakBeforeBraces: Custom
BreakConstructorInitializers: AfterColon
BreakConstructorInitializersBeforeComma: false
ColumnLimit: 110
ConstructorInitializerAllOnOneLineOrOnePerLine: false
ContinuationIndentWidth: 8
IncludeCategories:
  - Regex: '^<.*'
    Priority: 1
  - Regex: '^".*'
    Priority: 2
  - Regex: '.*'
    Priority: 3
IncludeIsMainRegex: '([-_](test|unittest))?$'
IndentWidth: 4
InsertNewlineAtEOF: true
MacroBlockBegin: ''
MacroBlockEnd: ''
MaxEmptyLinesToKeep: 2
SpaceAfterCStyleCast: true
SpaceAfterTemplateKeyword: false
SpaceBeforeRangeBasedForLoopColon: false
SpaceInEmptyParentheses: false
SpacesInAngles: false
```

```
SpacesInConditionalStatement: false
SpacesInCStyleCastParentheses: false
SpacesInParentheses: false
TabWidth: 4
AllowShortFunctionsOnASingleLine: None
PackConstructorInitializers: Never
AlignAfterOpenBracket: DontAlign
BinPackArguments: true
BinPackParameters: BinPack
BreakInheritanceList: AfterComma
ConstructorInitializerIndentWidth: 4
...
```