A Formal Requirements Engineering Toolkit

B. Eng. Project Example

Oliver Dixon

Department of Computer Science, University of York

Saturday, 20th June 2026

`https://github.com/oliverdixon/OptiFOL-Software`

Research Question

- Executing large engineering projects is difficult and error-prone, even for experienced professionals.
- Attaining a clear set of *project requirements* from the outset has been shown, over decades, to improve outcomes.
- But how do we know that our requirements are *useful*?
  - More precisely, how do we specify them?
  - And how do we know that other people know what we mean?
  - And if we have 100,000 requirements, how do we know that no two requirements are contradictory?
- These questions are hard to answer.
- So can we develop some software to make the process easier?

Research Answer

- We can use the language of *First-Order Logic* to formally represent requirements with no ambiguity.

  "The submarines should not be detectable by any sensors":

$$\underbrace{\forall x(\text{Submarine}(x))}_{\text{for all submarines...}} \implies \underbrace{\neg\exists y(\text{Sensor}(y)}_{\text{there is no sensor...}} \wedge \underbrace{\text{CanDetect}(y,x)))}_{\text{that can detect our sub!}}$$

- Once we have our requirements in first-order logic, we can reason about them using techniques from *Automated Theorem-Proving*.
  - Can they contradict one another?
  - Are any of them redundant?
  - Can they be made any simpler?

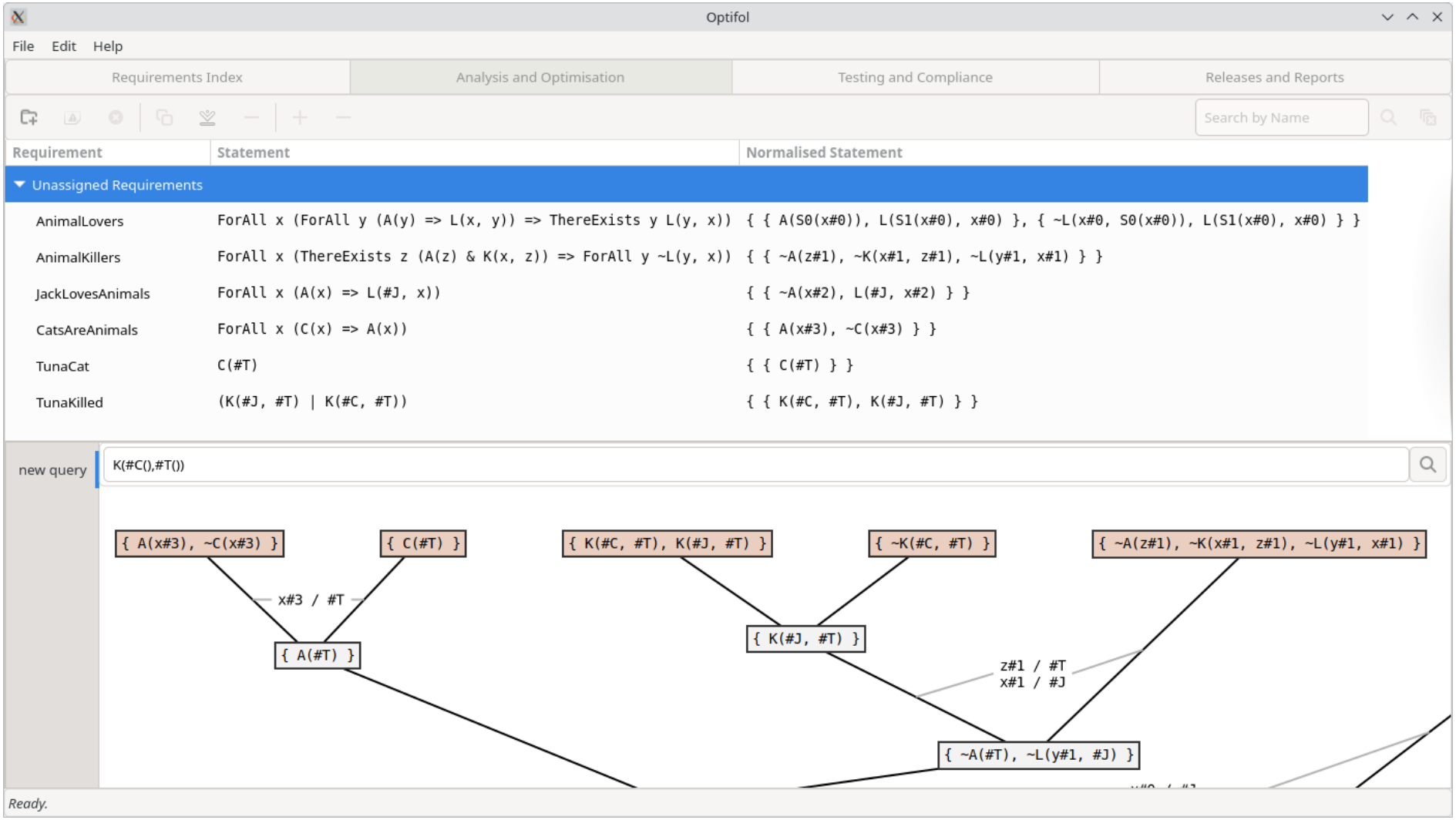
Project Outcomes

- Wrote approx. 35,000 lines of C++ to *prove* that this can be done.
- Produced a 61-page report (frontmatter; 30 pages of body; 15 pages of appendices; bibliography) describing the problem specification, design, implementation, and evaluation.
- Project marked at 81% by a team of world-leading experts in mathematical logic.
- Software has attracted interest from industrial partners.

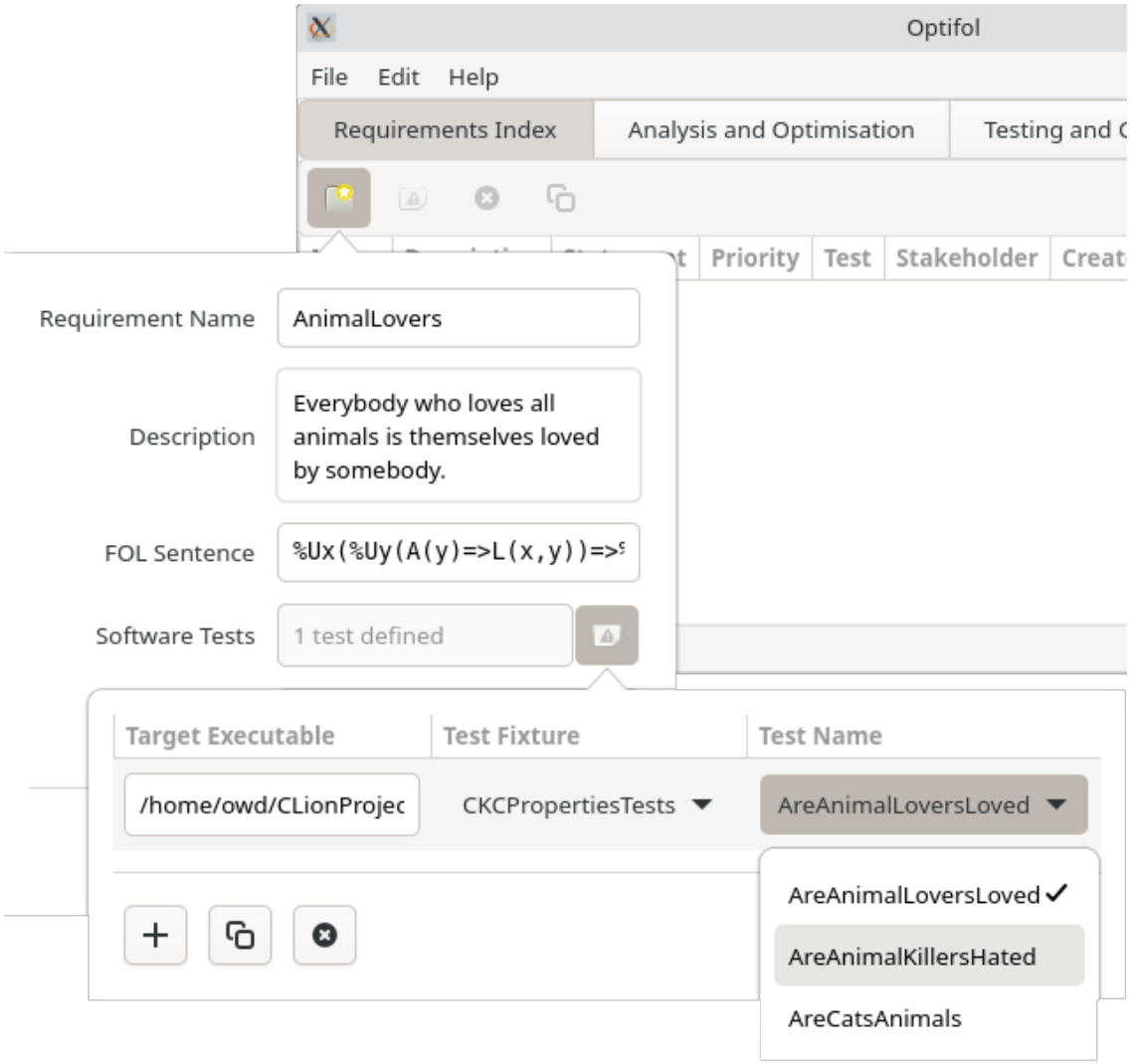
Optifol_

Inspiring
Confidence in
Engineering

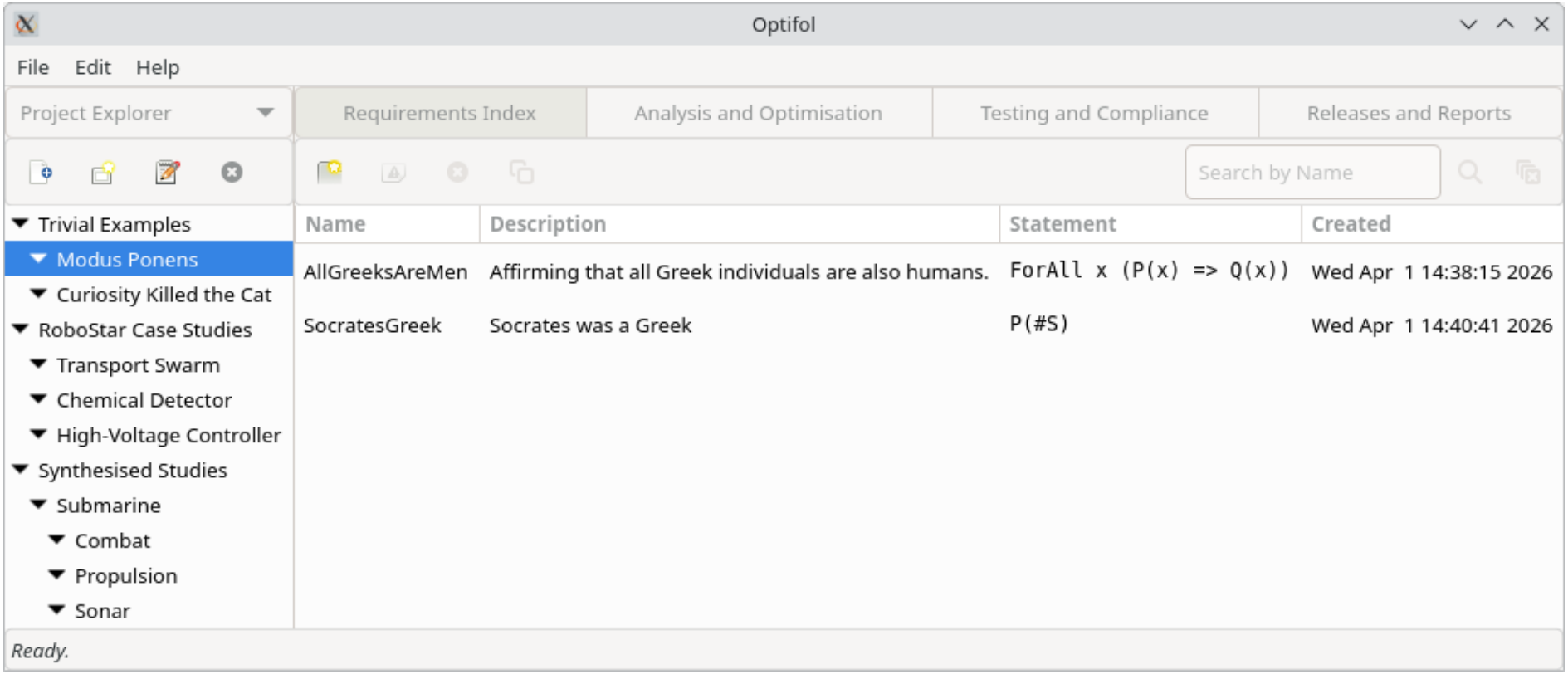
Visualising the Proof Procedure



Specifying a Requirement in Logic



Organising Requirements by Subsystem



Reviewing System Behaviour

Optifol

File Edit Help

Requirements Index | Analysis and Optimisation | Testing and Compliance | Releases and Reports

Search by Name

Name	Description	Statement	Created	Modified
ZeroVoltage	If the set-point is zero, the actual voltage is zero.	`ForAll x ((S(x) & I(x)) => (R(x) <=> Z(x)))`	Mon Apr 20 15:03:02 2026	Mon Apr 20 15:03:02 2026
PosVoltage	If the set-point is positive, the actual voltage is positive.	`ForAll x ((S(x) & I(x)) => (V(x) <=> P(x)))`	Mon Apr 20 15:04:20 2026	Mon Apr 20 15:04:20 2026
PWMZero	If the 24V power is off, PWM output is zero.	`ForAll x ((S(x) & I(x)) => (O(x) => W(x)))`	Mon Apr 20 15:05:18 2026	Mon Apr 20 15:05:18 2026
SetZero	If the 24V power if off, the set-point is zero.	`ForAll x ((S(x) & I(x)) => (O(x) => R(x)))`	Mon Apr 20 15:06:39 2026	Mon Apr 20 15:06:39 2026
Reach	All states are reachable	`ForAll x ((S(x) & I(x)) => E(x))`	Mon Apr 20 15:07:22 2026	Mon Apr 20 15:07:22 2026

Ready.

Adding Tests & Generating Qualification Reports