A Z Patterns Catalogue II

definitions and laws, v0.1

Samuel H. Valentine, Susan Stepney, and Ian Toyn

University of York Technical Report YCS-2004-383

October 2004

Contents

Preface **xiii**

I Background **1**

1 Introduction **3**

2 ISO Standard Z **5**
2.1 History and current status 5
2.1.1 Z: the early years 5
2.1.2 The need for a change 6
2.1.3 Standardisation 7
2.1.4 Aims of this chapter 7
2.2 Changed features of Standard Z 8
2.3 Improvements 8
2.3.1 Sections 8
2.3.2 Mutually-recursive Free Types 9
2.3.3 Operators 9
2.3.4 Conjectures 9
2.3.5 Binding Extensions and Tuple Selections 10
2.3.6 Schemas as Expressions 11
2.3.7 Empty Schemas 12
2.3.8 Loose Generics 13
2.3.9 Local Constant Declarations 13
2.3.10 Axiom-parts as Predicates 14
2.3.11 Soft Newlines 15
2.3.12 Lexis of Words 15
2.3.13 Toolkit 15
2.4 Incompatibilities 16
2.4.1 Singleton Sets 16

2.4.2 Decorated References to Schemas 17
2.4.3 Decorated References to Generic Schemas 18
2.4.4 let on Predicates 18
2.4.5 Renaming on Theta Expressions 18
2.4.6 Underlined Infix Relations 19
2.4.7 Operator Precedences 19
2.4.8 Theta Expressions 21
2.4.9 Lexis of Words 21
2.5 Subtle Changes 21
2.5.1 Quantified Expressions 21
2.5.2 Preconditions 22
2.5.3 Schema Instantiation 22
2.5.4 Precedence of lambda and mu 22
2.6 Conclusions 22

3 Patterns Catalogue Conventions 23
3.1 Background 23
3.2 Generalisations 24
3.3 Criteria for inclusion of definitions in the toolkit 25
3.4 Layout 26
Name of the pattern 26
3.5 Design 27
3.6 Presentation conventions 28
3.6.1 Specification style conventions 28
3.6.2 Distinct elements 28
3.6.3 Diagramming conventions 28
3.6.4 Global naming conventions 29
3.6.5 Local naming conventions 29
3.6.6 Consistency 30
3.7 Generic parameters 31
3.7.1 Implicit generic parameters 31
3.7.2 Generic schemas 31
3.7.3 Generic conjectures 32
3.8 Loose generics 33
3.8.1 Extendable definitions 34
3.8.2 Example: loose choice function 35
3.8.3 Example: polymorphic addition operator 35
3.9 Operator template paragraphs 36

II Core Language **39**

4 Core meta-language definitions **41**
4.1 Introduction 41
4.2 Syntax notation 41

5 Specification, Section, Paragraph **42**
Specification, Section 42
Paragraph 43
Simple declarations 44
Generic declarations 45
Conjectures 47
Operator templates 49

6 Predicates **54**
truth, falsity, and negation 55
Conjunction and disjunction 55
Implication 57
Equivalence 57
Universal quantification 58
Existential quantification 59
Unique existential quantification 61
Schema predicate 61
Relation predicate 62
Schema Text 64

7 Expressions **65**
Lambda expression 65
Mu expression 66
Let expression 67
Conditional expression 68
Cartesian tuple expression 68
Tuple component selection 69
Function and generic application expression 70
Reference expression 70
Set extension expression 71
Set comprehension expression 71

8 Schema Expressions **74**
Schema quantification 75

Schema propositional 77
Schema combination 79
Schema restriction 80
Schema renaming expressions 81
Schema construction 83
Schema binding construction 83
Schema binding extension 84
Binding selection 87

9 Type constructors 89
Given Set Paragraph 89
Free Type Paragraph 90
Power set Expression 94
Cartesian product Expression 96
Schema type 99

10 Lexis 102
10.1 Introduction 102
10.2 Z characters 102
10.3 Z words 103
10.4 Newlines and Line breaking 104

III Sets 105

11 Sets, Types and Values 107
11.1 Introduction 107
11.2 Z types 107
11.3 New sets 108
11.4 Set membership 108
11.5 Structure of the Part 109

12 Simple operations 110
Type signature 110
Boolean expressions 111
Negated core relation predicates 112
Cartesian square 113

13 Basic set operations 115
Empty Set 116

Set union 117
Set intersection 122
Set difference 126
Symmetric set difference 131
Distribution properties 133
Closure property 135

14 Subsets **137**
Subset 137
Proper subset 140
Non-empty subsets 141
Distribution property 142

15 Finiteness **144**
Finite sets 144

IV Binary relations **149**

16 Relations **151**
Tuple component selection 151
Relation notation 152
Finite relations 153
Maplet notation 154
Relational inverse 154
Deriving dual laws 156

17 Domain and Range of relations **158**
Domain and range 158

18 Relation restriction **161**
Domain restriction 161
Range restriction 166

19 Images, bounds, shadows **168**
Upper and lower image 168
Upper and lower bound 171
Upper and lower shadow 176
Upper and lower singleton image 180

20 Combining relations **182**

Compatible relations 182
Override 184
Composition 187
Demonic composition 192
Merge and split 195
Bicomposition 196

21 Functions 198
Functionality 199
General functions 200
Total functions 202
Finite functions 204
Surjections 204
Injections 207
Homomorphism 212
Isomorphism 214

22 Labelled Sets 217
Disjointness 218
Partition 219

23 Binary operators 220
Idempotence 221
Semigroup 222
Monoid 223
Group 225
Distributing an abelian monoid 228
Finite distributed sum 230
Finite distributed product 231

24 Homogeneous relations 233
Functions on the domain and range 236
Vertices 237
Roots and leaves 238
Identity relation 239
Reflexive relation 241
Reflexive closure 242
Irreflexive relation 243
Irreflexive residue 244
Symmetric relation 245

Symmetric closure and residue 246
Antisymmetric relation 247
Transitive relation 249
Transitive closures 251
Intransitive relation 253
Intransitive residue 255
Vertex finiteness 256
Equivalence relation 257
Acyclic relation 258
Maximal iteration 260

25 Connected graphs, forests and trees 263
Connected graph 263
Forest 265
Tree 267

26 Orders 269
Partial order 270
Poset 273
Total order 274
Chain 276
Preorder 277
A spectrum of orders 278
Minimum and maximum 280
Greatest lower bound, least upper bound 283
Well order 285
Well founded chain 286
Graph-preserving maps 287
Graph-reversing maps 290

27 Sorting 293
Sort 293

28 Two binary operators 297
Ring 297
Integral domain 300
Field 301
Ordered domain 302
Ordered field 303
Complete field 304

V Numbers **307**

29 Axiomatic Properties of Numbers **309**
Concrete syntax for number literals 310
Standard Prelude 311
Basic numerical operators 311
Integers and natural numbers 312
Rational numbers 314
Real numbers 316
Subtraction and division 317
Numerical orders 318

30 Further Numbers **319**
Sign 319
Absolute value 320
Floor and ceiling 321
Integer division and modulus 322
Integer range 324
Cardinality 326
Total cardinality 328
Minimum and maximum 330
Prime numbers 331
Square root 332

31 Numbers and Relations **334**
Relation iteration 334
Vertex degree 336

32 Extending to infinite sets **338**
Making a function complete on a set 338
Complete distributed sum 339
Complete distributed product 340

33 Powers and Trigonometry **342**
Factorial 342
Power function, integer exponent 343
Exponential function and natural logarithm 344
Power function, completed 345
Common logarithm 346
Sine and cosine 346

34 Streams and sequences **348**
General Streams 349
General Sequences 350
Finite streams 352
Finite sequences 353
Shifting the base of a stream 354
Stream and sequence displays 354

35 Constructing streams and sequences **355**
Concatenation 355
Enumerable order 357
Enumerable chains 359
Enumeration of an enumerable chain 360
Forming a sequence from a labelled set 360
Squashing a sequence from a numbered set 361

36 Prefix, suffix and infix orders **363**
Prefix and suffix relations 363
Infix relations 365
Prefix lower bounds 367
Prefix upper bounds 367
Suffix lower bounds 368
Suffix upper bounds 369
Infix lower bounds 370
Infix upper bounds 371

37 Manipulating streams and sequences **372**
Reversal 372
head, last, tail, front 374
Extraction and filtering 377
Paths and steps 380

38 Sequenced families of sets **384**
Distributing a monoid over a sequence 385
Distributed override 386
Distributed composition 387
Distributed concatenation 388

39 Bags **390**
Bags 390

Functions of a single bag 392
Functions of two bags 392
Building a bag 394
Bag display 395
Further bag operations 396

VI Example Specifications 401

40 Changing representations: a memory map 403

41 Neural networks 405
41.1 Introduction 405
41.2 A network 405
41.2.1 A simple network 405
41.2.2 A feed forward network 406
41.2.3 A layered network 407
41.3 Pattern Recognition 407
41.4 Net Training 407
41.5 Further Reading 408

42 Kinship 409
42.1 Introduction 409
42.2 Ancestors 409
42.3 The royal house of Thebes 411
42.4 The ancestral line 412
42.5 Matrilineal and patrilineal relations 414
42.6 Specialisations of the ancestor relation 416
42.7 Descendants 417
42.8 Collateral relations 417
42.8.1 Disjoint lines 418
42.8.2 Common ancestors 418
42.9 Siblings 419
42.10 Aunt, uncle, niece, nephew, cousin 420
42.11 General theorem of blood relationship 423
42.12 Antigone's relations 424
42.12.1 Antigone's grandparents' grandchildren 424
42.12.2 Antigone's great-grandparents' great-grandchildren 425
42.12.3 Summary 426
42.13 Monotonic functions on sets 426

42.14Other compositions 427
42.14.1 Composition of descendant with ancestor 427
42.14.2 Composition with collateral 428
42.15Enriching the model 428
42.15.1 Including birth order 428
42.15.2 Including relationship by marriage 428
42.15.3 Relaxing the axiom of ancestry 429
42.16Further reading 429

43 Chess 431
43.1 Introduction 431
43.2 The board 431
43.3 The chessmen 432
43.4 Board positions 432
43.5 The starting position 433
43.6 Moves in general 433
43.7 Moves in particular 436
43.8 Legal moves 440
43.9 Winning 441
43.10Draws 441
43.11Other endings 443
43.12Optimal play 443

VII Appendix 445

A Diagrammatic conventions 447
A.1 Introduction 447
A.2 Venn diagrams 447
A.3 Diagrams for relations 448
A.3.1 Venn diagrams 448
A.3.2 Cartesian diagrams 449
A.4 Functions 450
A.5 Homogeneous relations 450
A.5.1 Venn diagrams 450
A.5.2 Cartesian diagrams 451
A.6 Orders 451
A.6.1 Cartesian diagrams 451
A.6.2 Network diagrams 452

B Summary of operator templates **453**

C Bibliography **458**

D Proofs **464**

E Index **493**

Preface

The Z Patterns Calatogues

The various volumes in the *Z Patterns Catalogue* series, outlined below, are evolving documents – as we discover and are informed of more patterns, we will add them to new versions of the catalogues.

The three catalogues, history and plans for the future

- **I : specification and refactoring** (Stepney, Polack, Toyn)
  - v0.1 (YCS-2003-349): The initial structure, with a focus on promotion as a generative pattern, and refactoring, with many skeleton patterns (particularly in the developmental section)
  - v0.2 (planned early 2005): fleshed out skeletons, more patterns, and material from *Z in Practice*
- **II : definition and laws** (Valentine, Stepney, Toyn)
  - v0.1 (this catalogue): The initial structure, of a rich mathematical toolkit
  - v0.2: More generic patterns, including a type-constrained generic schema toolkit, and patterns for generating toolkits by abstraction
- **III : proof and refinement** (Cooper, Stepney, Woodcock)
  - v0.1 (planned end 2005): The initial structure, with proofs of interesting properties, and refinement as a generative proof pattern
  - v0.2: Refactoring proofs, retrenchment as 'approximate proof refactoring'

Acknowledgements

Many people have contributed in a number of ways to this catalogue, commenting on various drafts over the years, suggesting examples to specify, and discussing

approaches to specification. Our thanks to you all, including Rob Arthan, Tim Ball (for proving some of the laws here, and disproving others, which were subsequently removed), Jane Gardner, Ian Hayes, Helen King, Trevor King, Fiona Polack, Dan Simpson, Alf Smith, Jim Woodcock, John Wordsworth.

Our thanks also to Visio, Donald Knuth for TEX, Leslie Lamport for LATEX, Mike Spivey for the original Z Mathematical Toolkit, and Paul King for `oz2e.sty`, without whom this catalogue would have looked very different.

Part I

Background

Chapter 1

Introduction

> *It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilisation advances by extending the number of important operations which we can perform without thinking about them. Operations of thought are like cavalry charges in a battle — they are strictly limited in number, they require fresh horses, and must only be made at decisive moments.*
>
> — Alfred North Whitehead, *An Introduction to Mathematics*

Why a bigger toolkit?

When writing a Z specification, we want to capture those things that are essential properties of the system, and omit those things that are merely contingent. We want to have as much structure as is necessary, but no more. For example, we should not model something as a sequence if we merely want to label the elements, if their order is unimportant.

Our historical heritage is Spivey's Mathematical Toolkit [Spivey 1992]. Spivey's toolkit gives us a language for modelling. In this catalogue we extend that language to a richer vocabulary that enables us to make finer discrimination of the essential properties.

The possibility of abstraction is often suggested when the same or closely similar formal text occurs more than once in a specification. It may then be useful to separate out the common material as one or more schemas, functions, sets and so on whose properties and relationships can then be explored and proved in isolation, away from the context of use. Doing this can make specifications as a whole briefer. It can also suggest useful generalisations. The decision as to whether to do it should

be taken above all on the basis of maximising correctness and clarity.

In this catalogue we have made some Z definitions and developed some theories, all of which correspond to mathematical entities that have been known about for many years. Familiarity with this material will allow specifiers to make use of that established understanding without having to reinvent it. Our experience has been that many of the structures presented here are useful in specification work. Our aims in writing this catalogue are both to provide useful general theories and to encourage users to develop their own.

We are not doing anything here that could not be done in your own specification. However, having a rich pre-exisiting vocabulary can be a great advantage: it helps you to distinguish what may be the essential properties; it reduces the need to keep reinventing subtly different wheels; it reduces the size of the specification you need to write; it reduces the cognitive load for a reader already familiar with the larger vocabulary.

Granted, it may take more time to become familiar with this rich vocabulary initially, but it can be worth the effort in the long run. This is a similar theme to the use of a class library provided with an object-oriented programming language such as Smalltalk or Eiffel: it takes a while to learn such a library, but it gives you greater power in the end.

We have structured the catalogue using our notation for Z patterns [Stepney *et al.* 2003].

Chapter 2

ISO Standard Z

Indulging myself in the freedom of epistolary intercourse, I beg leave to throw out my thoughts, and express my feelings, just as they arise in my mind, with very little attention to formal method.

— Edmund Burke, *Reflections on the Revolution in France*, 10th edition, 1791.

2.1 History and current status

2.1.1 Z: the early years

In the early days of Z specification and publication, authors were obliged to include some kind of 'Z appendix', to explain the language they were using. As the language has matured, fuller descriptions have become available.

- [Abrial 1980] *The Specification Language Z* was one of the earliest documents on Z.
- [Sufrin *et al.* 1984] *The Z Handbook* was one of the first descriptions of Z available to the wider community.
- [Hayes 1987] *Specification Case Studies*, contains its own Z appendix, including a summarised mathematical toolkit. The bulk of the book comprises many excellent examples of typical ways of using Z to build specifications. It is currently in its second edition [Hayes 1993] with syntax brought more closely in line with *ZRM* (see below).
- [King *et al.* 1988] *Z: Grammar and concrete and abstract syntaxes*, also known as the *Yellow Book*, from the colour of its cover, gives one of the first formal description of Z's syntax.
- [Spivey 1988] *Understanding Z*, also known as the *Blue Book*, describes semantics for (a subset of) Z.

- [Spivey 1989] *The Z Reference Manual*, (*ZRM*), gives a full syntax description for Z, an informal semantics, and a Z specification of the *Mathematical Toolkit*. It too is currently in its second edition [Spivey 1992] with let, if, schema piping and more bag operators added.

There are also many publications on how to use Z: some tutorials for beginners, for example [Potter *et al.* 1991] *An Introduction to Formal Specification and Z*; some for more advanced users, for example [Barden *et al.* 1994] *Z in Practice*; some emphasising refinement, for example [Wordsworth 1992] *Software Development with Z*; some emphasising proof [Woodcock & Davies 1996] *Using Z*. These tend to follow *ZRM*.

2.1.2 The need for a change

Because of these multiple descriptions of Z, the need for a *Standard* Z has been felt for some time. This need has become more pressing as tool-support for Z has grown. Where dialects differ, which should a tool support? Where ambiguities exist, what resolution should a tool adopt?

This need is felt most sharply as the use of *proof* in Z increases. For example, when performing a proof, a common step is to expand an expression by replacing a name by its definition. One might wish to replace the name of a schema with an expression that defines its value. However, in *ZRM*, such a substitution is not syntactically valid: an Expression may be a Schema-Ref, but not a Schema-Exp. If the syntax were more liberal, to allow 'substitution of equals for equals' to be syntactically valid everywhere, then proofs could be performed more easily.

In addition to syntax problems with proof, it has become clear that the semantics described in the *Blue Book*, although an excellent first step, is not in a form suitable for a Standard defintion. It is the published form of Spivey's doctoral dissertation, which was written to support the thesis that giving a satisfactory semantics for Z is a soluble problem; this it successfully does. A Standard, however, needs to be organised in a different way from a dissertation, with everything spelt out explicitly and in a way suitable for reference purposes.

Z has become mature enough that others wish to use it to define their own international standards. The Open Distributed Computing (ODP) community, in particular, felt that their own work could benefit by being formally specified in Z. ISO rules require that for it to be used in another standard, Z must be a standard itself.

2.1.3 Standardisation

The main guiding principles for Z's standardisation are

- to remove certain unnecessary restrictions (in particular, to liberalise the syntax)
- to define the semantics (type rules and meaning)
- to remove the need for certain special cases, to allow things to be defined in the Toolkit, rather than be hardwired into the base language
- to allow correct existing Z specifications to remain correct *Standard* Z specifications, as far as possible

The process of standardising Z began under the ZIP project, part-funded by the United Kingdom's Department of Trade and Industry, initially for submission to the British Standards Institute (BSI). After the end of that project, the standardisation work continued, under the banner of the International Standards Organisation (ISO). Z became an ISO standard in 2002 [ISO-Z 2002].

2.1.4 Aims of this chapter

There are many excellent tutorials for *ZRM*, and *Standard* Z is mostly a superset of that, with a few small changes. Here we highlight the main differences between *ZRM* and *Standard* Z, especially those that we exploit in writing the rest of this catalogue.

The authors of this catalogue were all members of the Z Standards Panel, and many of the ideas explained here can be attributed to other members of the Z panel. The Z panel has taken into account other work from the history of Z besides *ZRM*, including those early works mentioned in section 2.1.1. The formal reasoning experiences gained with ProofPower [Arthan 1991], Zola [Harwood 1995] and CADiℤ [Jordan *et al.* 1991] [Toyn 1996] have had some influence. Most users of Z regard *ZRM* as their *de facto* standard, and so that is the most suitable point of comparison for this chapter. For some historical context for these ideas, see [Toyn 1998], from which the remainer of this chapter has been adapted and updated.

There is a lot more that could be said about the Standard document and its history, and maybe one day some historian will do so.

2.2 Changed features of Standard Z

ZRM was a huge step forward at the time of its first publication. Many syntax and typechecking tools for Z have been based on it, and it has become a *de facto* standard. It is an excellent work that has certainly served the Z community well. However, there are several issues that it does not adequately address. Standardisation aimed to provide widely acceptable solutions to the issues, so that diverse dialects can be avoided.

The rest of this chapter is in three parts. First, the restrictions of *ZRM* are identified and *Standard* Z's solutions are outlined. Second, the incompatibilities arising from those solutions between *ZRM* and *Standard* Z notations are discussed. Third, some subtle changes are explained where, although existing *ZRM* notation still has the same meaning, the interpretation of the notation to give that meaning has changed.

2.3 Improvements

This section presents improvements in *Standard* Z that address inadequacies in *ZRM*.

2.3.1 Sections

Specifications are rarely written in terms of the Z base language. Even pedagogic examples usually refer to the definitions of the mathematical toolkit. Real specifications are constructed from libraries or toolkits of operations relevant to particular application domains. It should be possible to reuse toolkits by reference, without having to duplicate them into every specification that uses them. This is an issue that *ZRM* ignores. Toolbuilders usually incorporate *ZRM* mathematial toolkit by default, which causes problems for specifiers who want a different toolkit.

Standard Z provides probably the simplest possible solution to the toolkit reuse problem, in the form of its section notation.

The mechanics of how sections are brought together is unspecified. Sections might be separate documents, or chapters or appendices in the same document. A tool might use a relation between section names and file names.

2.3.2 Mutually-recursive Free Types

ZRM presents a free type as an abbreviation for a given type followed by axiomatic constraints to ensure that its constants are elements of the given type, that its constructors are injections producing members of the given type, that the elements and the values returned by the injections are all distinct from each other, and that all values of the type are either elements or returned by an injection.

Standard Z extends *ZRM* free types to allow mutually-recursive free types. The mutually-recursive free types are written within a single paragraph, the parts separated by & characters.

2.3.3 Operators

An operator is a name with special lexical status, for example an infix operator appears between its operands. *ZRM* notation allows the use of various operators. A use of an operator has to be preceded by its definition. Its definition ought to be preceded by some kind of *template* for the operator, to indicate that the name will be defined and used in, for example, infix position. Without the information provided by such a template, it is not possible to parse the operator's definition and uses. *ZRM* says what kinds of templates it permits operators to have, but it specifies no notation for the introduction of operator templates. It presumes that all the operators defined in its toolkit are already known to the reader and hence recognisable, and leaves each tool to implement its own distinct notation for templates.

Standard Z has an *operator template paragraph* that serves to introduce new operators.

2.3.4 Conjectures

ZRM presents many laws, particularly about the toolkit operators, but without formalising their syntax as part of Z. Their presentation is pseudo-formal, none of their variables being declared.

Proof tools typically provide *sequents*, which are a notation suitable for expressing not only laws, but also conjectures, theorems, goals, lemmas and axioms. Of these, conjectures are the starting point for proofs, and are sometimes hand-written within specifications. Different proof tools use different syntaxes for sequents, and it was not thought appropriate to standardise their syntax. However, standardising a simpler syntax specifically for conjectures is possible and worthwhile, as this

	ZRM		*Standard* Z	
	Constructors	Selectors	Constructors	Selectors
Tuples	(x, y, z)		(x, y, z)	$triple.3$
Bindings		$binding.name$	$\langle\!\langle\ x == 1, y == 42\ \rangle\!\rangle$	$binding.name$

Table 2.1 Product type operations

allows them to be written within specifications in a form that potentially eases their interchange between tools and allows them to be subjected to typechecking.

Standard Z's notation for a (generic) conjecture involves (a generic parameter list,) a $\vdash?$ symbol, and a single predicate. This simple syntax is chosen as it is likely to conform to the syntax of a sequent, or at least be translatable to a sequent, whatever proof tool is used.

In this catalogue, we use an extended conjecture syntax; see §5, Conjectures : variant for details.

2.3.5 Binding Extensions and Tuple Selections

The Z base language provides both labelled and unlabelled product types, called schema types and Cartesian product types respectively. One would expect to find notations for construction and selection operations on values of each of these types, but *ZRM* offers only a selection operation for values of schema type, and only a construction operation for values of Cartesian product type (table 2.1). *Standard* Z also offers notations for the other two operations.

ZRM explains bindings using the notation $\langle p_1 \Rrightarrow x_1,\ ...,\ p_n \Rrightarrow x_n \rangle$ (pages 26 and 62), but does not permit use of this as Z notation. *ZRM* notation is used largely for producing abstract specifications of systems, where the emphasis is on the use of schemas and constraints on them rather than particular bindings, those being more specific and concrete. However, Z can be used in other ways and in other contexts, for example, in reasoning about a relational database, where the rows of a table could be modelled by the bindings of a schema. Bindings can arise in *ZRM* either by theta expressions or as members of schemas. It is particularly useful to have such notation during proofs, where showing the truth of predicates such as $\theta S = \theta S'$ involves consideration of the underlying binding values. (An example of such a proof appears in the next subsection.)

Standard Z's notation for the construction of bindings from explicit component values is called a *binding extension expression*: $\langle\!|\ i_1 == e_1,\ ...,\ i_n == e_n\ |\!\rangle$

ZRM notation provides *first* and *second* selectors for pairs in the toolkit, but no selectors for larger tuples. *Standard* Z's notation for the selection of components from tuples is called a *tuple selection expression*: $expr.n$. This allows one to say things such as $(x, y, z).2 = y$. *Standard* Z retains *first* and *second* in the toolkit for backwards compatibility.

2.3.6 Schemas as Expressions

An expression has a value of a particular type. A schema has a value — it is a set of bindings — but *ZRM* syntax permits only references to named schemas, not general schema expressions, to be used as expressions. Instead, *ZRM* has a separate category of schema expressions, that may appear only in named horizontal schema definition paragraphs. This distinction between schema expressions and other expressions also prohibits an expression whose type is that of a set of bindings from being used within a schema expression.

This syntactic restriction can result in some added verbosity in specifications, but its main problem is that it is an obstacle to formal reasoning. The replacement of a name by its defining expression is a typical 'substitution of equals for equals' logical inference, but is precluded by the syntactic restriction. Without that particular inference rule, it is not clear how to replace a reference to a schema by the mathematics of its definition, and hence to reason further. More generally, all formulae arising from logical inferences should be expressible in the concrete syntax, and so irregularities in the concrete syntax should be eradicated.

In *Standard* Z, the syntactic category of schema expressions is merged into that of expressions. So an arbitrary schema expression may appear wherever a schema reference could appear in *ZRM* notation: as an inclusion declaration, as a predicate, as an operand to θ, or as an expression. The type system ensures that a schema is used only where an expression whose type is that of a set of bindings is permissible, and that only an expression whose type is that of a set of bindings is used where a schema is required.

Tutorials on Z can now give a much simpler description of schemas, for example:

> A schema is any value whose type is a set of bindings. In addition to its ordinary use as a set, a schema may be used in three special, and important ways: (i) as a declaration; (ii) as a predicate and (iii) as an operand of certain

> special operators (called the schema calculus operators) which construct new schemas from old in various convenient ways.
>
> — [Jones 1992]

The following example illustrates use of schema expressions as inclusion declarations and as operands to θ.

$$S == [\ x : \mathbb{Z};\ y : \mathbb{N}\ ]$$
$$\Delta Sx == [\ S;\ S'\ |\ \theta(S \setminus (x)) = \theta(S \setminus (x))'\ ]$$

(See §2.3.9 for an explanation of the use of == rather than $\widehat{=}$ when declaring these schemas.)

The ΔSx schema can be interpreted as defining a change to the state represented by schema S in which only the x component's value can change. The conjecture that the value of component y is left unchanged by ΔSx can now be stated and proved. In this proof, schema expressions can be seen being used as predicates. (See §5, Conjectures : variant for details of the variant conjecture syntax used here.)

$\vdash\ \forall \Delta Sx \bullet y = y'$
$\vdash\ \forall S;\ S' \bullet \Delta Sx \Rightarrow y = y'$ expand ΔSx
$\vdash\ \forall S;\ S' \bullet [\ S;\ S'\ |\ \theta(S \setminus (x)) = \theta(S \setminus (x))'\ ] \Rightarrow y = y'$ expand ΔSx
$\vdash\ \forall S;\ S' \bullet [\ S;\ S'\ |\ (\!|\ y == y\ |\!) = (\!|\ y == y'\ |\!)\ ] \Rightarrow y = y'$ expand θs
$\vdash\ \forall S;\ S' \bullet [\ S;\ S'\ |\ y = y'\ ] \Rightarrow y = y'$ absorb binding extensions
$\vdash\ \forall S;\ S' \bullet y = y' \Rightarrow y = y'$ absorb schema predicate
$\vdash\ \forall S;\ S' \bullet true$ absorb implication
$\vdash\ true$ absorb universal quantification
■

2.3.7 Empty Schemas

An empty schema is a schema with no declarations. One can arise in *ZRM* notation from the hiding of all declarations from a schema.

$$Schema == [x, y : \mathbb{Z} \mid x \neq y] \setminus (x, y)$$

This should simplify to the following equivalent paragraph:

$$Schema == [\ \mid \exists x, y : \mathbb{Z} \bullet x \neq y]$$

ZRM does not permit this to be written, as the list of declarations is not allowed to be empty. *Standard* Z does allow this.

2.3.8 Loose Generics

ZRM informally requires that generic definitions not be loose:

> A restriction must be obeyed by the definitions of generic constants for them to be mathematically sound: the definition must uniquely determine the value of the constant for each possible value of the formal parameters. ... [This] places a proof obligation on the author of a specification ...
>
> — [Spivey 1992, §2.4]

That proof obligation is rarely discharged by authors.

The reason for this restriction is discussed in [Spivey 1988, §4.1.1]. *Standard* Z relaxes this restriction, which allows us to use *loose generic definitions*; we no longer have to uniquely define each generic constant. However, it is important to ensure that a generic is not loose when it is used:

- either, provide supplementary constraints to tighten it before it is used (although such 'nonconservative extension' might cause difficulties for some proof tools)
- or, use only the part of its definition where it is not loose (for example, a generic function may be loose on part of its domain, if it is used only on parts that are not loose)

2.3.9 Local Constant Declarations

ZRM restricts the use of == to the global level in non-generic declarations and to (2nd edition) let declarations. This is an unnecessary irregularity. *Standard* Z removes this restriction, and allows such an 'equality declaration' to be used anywhere a 'colon declaration' is valid: the declaration $x == e$ is equivalent to the declaration $x : \{e\}$. This form of declaration has the advantage that the uniqueness of the value of x is clear.

As a conseqence of merging schemas and expressions in the syntax, declaring a schema S to be some set of bindings is just a special case of declaring x to be some

set of values. So *Standard* Z uses the same notation for these declarations:

$$S == [\ x_1, \ldots x_n : X \mid P\ ]$$
$$U == S \land T$$
$$x == \{\ x_1, \ldots x_n : X \mid P\ \}$$
$$y == 42$$

So *Standard* Z does not use the '$\widehat{=}$' symbol for schema declaration.

When used in quantified predicates or definite description (or let) expressions, local constant declarations provide a neat notation for expressing substitutions.

$$\exists\, x == 42;\ y == 1998 \bullet p$$
$$\mu\, x == 42;\ y == 1998 \bullet e$$
$$\textbf{let}\, x == 42;\ y == 1998 \bullet e$$

This generalisation makes the let notation redundant, but it is retained for backwards compatibility with *ZRM* (although only the expression form is retained, as explained in section 2.4.4).

2.3.10 Axiom-parts as Predicates

ZRM's 'Predicate' paragraph is not present in *Standard* Z; since the declaration part of an axiomatic definition may be empty, a predicate constraint can be written as

$$\vdash \mathcal{P}$$

Alternatively, if the predicate is not meant as a constraint, but is rather the statement of some property of the specification, it can be written as a conjecture (see §2.3.4 and §5).

ZRM permits newline or semicolon to separate outermost conjuncts in an Axiom-part. *Standard* Z removes the unnecessary irregularity that distinguishes Axiom-parts from other predicates by permitting newline or semicolon between any predicates to mean conjunction, and giving newline and semicolon very low precedences, so that any such new uses of newline and semicolon must be parenthesized. So the following are permitted in *Standard* Z, and are equivalent:

$$\begin{array}{|l} \ldots \\ \hline \exists\, S \bullet \\ \quad \mathcal{P};\ \mathcal{Q} \end{array} \qquad \begin{array}{|l} \ldots \\ \hline \exists\, S \bullet \\ \quad \mathcal{P} \\ \quad \mathcal{Q} \end{array} \qquad \begin{array}{|l} \ldots \\ \hline \exists\, S \bullet \\ \quad (\mathcal{P}) \\ \quad \land\, (\mathcal{Q}) \end{array}$$

2.3.11 Soft Newlines

Newlines serve two different purposes in Z: *hard newlines* separate declarations and conjuncts; *soft newlines* merely break up long formulae onto multiple lines without themselves having any semantic significance. In *ZRM*, newlines are soft if they are adjacent to an infix operator. In *Standard* Z, newlines are also soft if they follow a prefix operator or precede a postfix operator. This recognises the other circumstances where the next line must contain a continuation of the same formula. A particular case that was not breakable in *ZRM* is the application expression

$$veryLongFunctionExpression\ veryLongArgumentExpression$$

which in *Standard* Z, with the addition of parentheses, can be broken.

$$veryLongFunctionExpression(\\ veryLongArgumentExpression)$$

2.3.12 Lexis of Words

In the *ZRM* syntax, Word is a terminal symbol. There is some informal description of a Word being made up of letters, digits, underscores and other symbols, but it is imprecise about what those are and how they can be put together. *Standard* Z gives a formal definition of Word, referring to the symbols of the Unicode standard [Unicode 1996]. It is a very flexible definition, and largely compatible with traditional practice.

2.3.13 Toolkit

A major inadequacy in *ZRM*'s toolkit is the omission of a formal definition of the numeric operations. This omission is resolved in *Standard* Z as follows. The integers $\mathbb{Z}$ are replaced as the basis for numeric operations by the set $\mathbb{A}$ (pronounced "arithmos"), representing an unrestricted concept of number; $\mathbb{A}$ is introduced in the prelude section. The prelude section also introduces $\mathbb{N}$, 0, *succ*, and addition and multiplication of naturals, which gives the minimum necessary to provide a basis for the semantics of natural number literals in the Z base language. The prelude section is written in Z like any other section, but is regarded as part of the Z base language for the purpose of the semantic definition; it is an implicit parent of every other section.

The integers $\mathbb{Z}$ and further properties of the natural numbers $\mathbb{N}$ are defined in the Standard toolkit. Having these sets as subsets of $\mathbb{A}$ and making the numeric

operations be partial functions on $\mathbb{A}$ avoids having either to introduce distinct names for the operations on different subsets or to introduce an overload resolution mechanism into Z. (The *ad hoc* overloading introduced by loose generics is inappropriate in this context.) The numeric functions and relations declared in [Spivey 1992, §4.4], except *succ*, have had their domains widened where appropriate, while retaining their *ZRM* meanings for integers. This allows their definitions to be widened to cope with other kinds of numbers, such as reals $\mathbb{R}$. We have availed ourselves of this opportunity in this catalogue.

The Standard toolkit also revises the *ZRM* toolkit in other small ways. The meanings of most definitions remain unchanged, though the domains of some are widened.

2.4 Incompatibilities

This section lists backwards incompatibilities between *ZRM* and *Standard* Z arising from the changes discussed in the previous section. For each incompatibility is given an explanation of *what* it is, a rationale for *why* it exists, and some notes on *how* instances of it can be detected and rectified.

2.4.1 Singleton Sets

The notation $\{i\}$, where i is the name of a schema, is parsed differently: *ZRM* parses it as a set comprehension, whereas *Standard* Z parses it as a singleton set extension. (In *ZRM*, the set extension (set display) containing a single schema reference is written $\{(i)\}$.)

Standard Z permits any schema-valued expression to be written wherever *ZRM* permits only a schema reference, so the potential ambiguity between singleton set extensions and set comprehensions is broadened to expressions matching the pattern $\{e\}$. Since e can contain parentheses, the ambiguity cannot be resolved in the way *ZRM* resolves it. (Where e is not a schema name, both parse $\{e\}$ as a set extension.)

The type of the *ZRM* set comprehension $\{i\}$ is that of a set of bindings, whereas the type of the *Standard* Z set extension $\{i\}$ is that of a set of sets of bindings, so a typechecking tool will detect and report most instances of this incompatibility. The *Standard* Z coercion to a set comprehension is to write $\{i \mid true\}$. The value of the set comprehension is just i, and that equivalence holds in *ZRM* notation too, so another translation from *ZRM* to *Standard* Z is just to drop the set brackets.

2.4.2 Decorated References to Schemas

Any decoration on a reference to a schema must be separated from the schema name.

The merging of schemas and expressions has included the merging of the name-spaces of schema names and other names, so a schema can now be defined with a decoration within its name. For each expression comprising a name with a decoration, there are two possible intentions and hence interpretations: either the name refers to a schema declaration in the environment and the decoration is to be applied to the components of that schema, or the decorated name refers to a schema declaration in the environment whose name is itself decorated. *Standard* Z must be able to express either intention, whereas *ZRM*-compliant specifications have only the former possibility.

Standard Z distinguishes the two intentions by the presence or absence of separation between the name and decoration. That separation can be either white space or parentheses around the name. For example, consider the parentheses in the following.

$$S == [\ x : \mathbb{N}\ ]$$
$$S' == [\ y : \mathbb{N}\ ]$$
$$T == (S)' \land S'$$

The expression $(S)'$ is a reference to schema S with its components decorated, so $(S)' = [\ x' : \mathbb{N}\ ]$, whereas the expression S' is a reference to the schema S'. So $T = [\ x', y : \mathbb{N}\ ]$. The decoration expression $(S)'$ could equally be written $S\ '$ (note the space).

Separation is needed to get the *ZRM* interpretation, but *ZRM* specifications might not have that separation. If there is no white space, then a type-checking tool will report that the decorated schema name is not declared.

This backwards incompatibility could be avoided by transforming undeclared decorated references to decoration expressions, distributing the minimum number of strokes from the references to the decoration expressions to give a type correct result. However, reliance on this transformation in new specifications would make those specifications less clear for readers, and the transformation would complicate both tools and the standard. Use of separation in decoration expressions should be at least encouraged.

2.4.3 Decorated References to Generic Schemas

The decoration and instantiation on a reference to a generic schema is reversed, for example, *ZRM*'s $S'[\mathbb{N}]$ becomes in *Standard* Z $S[\mathbb{N}]'$.

ZRM treats both the decoration and the instantiation as part of a schema reference, requiring them to be written in that order. *Standard* Z requires the instantiation to be on the reference, but the decoration could be on any schema, and so the decoration must follow the instantiation.

A syntax checking tool will recognise a decoration expression, and will then be able to recognise an instantiation list (that being distinct from a schema construction expression, and can appear only in different contexts to generic parameter lists), but will be unable to recognise their juxtaposition, so a syntax error is guaranteed. The decoration and the instantiation must be reversed.

2.4.4 let on Predicates

The let notation introduced in *ZRM* cannot be used as a predicate in *Standard* Z.

In *ZRM*, in a context where a predicate is expected, a let with a schema name after its $\bullet$ can be parsed as either a let expression used as a predicate or as a let predicate with a schema name used as a predicate, but both have the same meaning. In *Standard* Z, any schema expression can be used after the $\bullet$, and so there can be free variables in that part, leading to different meanings depending on whether the let is taken to be an expression or a predicate. So *Standard* Z cannot have both let expressions and let predicates. Neither is needed, thanks to local constant declarations. The expression form is retained, as it allows some uses of μ, less familiar to non-specialists than $\exists$, to be avoided.

A syntax checking tool will detect some uses of let on predicates, but might mistake some uses of let on relational predicates as let on the leading expression. Each use of let on a predicate should be replaced by $\exists$ (or by $\exists_1$ or $\forall$ since all mean the same given that the quantified declarations are all $==$ declarations).

2.4.5 Renaming on Theta Expressions

The square-bracketted renaming notation on theta expressions, that was introduced in *ZRM*, is parsed differently in *Standard* Z.

The Z Standard's merging of the syntaxes of schemas and expressions has resulted in a single schema renaming production that permits renaming of any expres-

sion, with the type constraint that the expression be a schema. The renaming production has lower precedence than that of θ, so $\theta S[new/old, \ldots]$ is parsed as $(\theta S)[new/old, \ldots]$.

Since θS is a binding, not a schema, and renaming is permitted only of a schema not a binding, the renaming of θS will always be detected as a type error. The *ZRM* $\theta S[new/old]$ denotes the binding $\langle\!|\ old == new\ |\!\rangle$. (It is interesting to note that the effect of the notation is not to rename the name on the left of the == but rather to substitute for the value on the right.) That same binding can be built in *Standard* Z using the notation **let** $old == new \bullet \theta S$ (which with the addition of surrounding parentheses is valid in *ZRM* too).

2.4.6 Underlined Infix Relations

The underlining notation for infix relational operators, introduced in *ZRM*, cannot be used in *Standard* Z.

In *ZRM*, there is no way of introducing new operator notation, so instead each use of an identifier as an infix relation can be underlined to make clear that it is being used as an infix symbol. In *Standard* Z, operator template paragraphs provide a way of introducing new operator notation, so there is no need to mark uses of it as such. Moreover, the underlining notation has not caught on, and it does not help with operators other than infix relations.

A syntax checking tool will detect all uses of underlining notation. Each underlined infix relational operator should be declared in an earlier operator template paragraph.

2.4.7 Operator Precedences

Table 2.2 enumerates the relative precedences of the predicate and expression notations in *ZRM* and *Standard* Z, from lowest at the top to highest at the bottom, revealing some differences. Schema expression notations of *ZRM* are omitted, as they appear in separate contexts. The relative precedences of *ZRM*'s schema calculus operations are, from lowest to highest, $\gg$, $\fatsemi$, $\setminus$, $\upharpoonright$, $\Leftrightarrow$, $\Rightarrow$, $\vee$, $\wedge$, pre, $\neg$.

Operator templates cause several rows of the table to appear to be different, but in fact the only change in relative precedence caused by them is that between *juxtaposed function applications* and *postfix functions*. The merging of schema expressions with expressions is the cause of most of the differences. Many schema expressions use the same operators as quantified or logical predicates. In *Standard*

ZRM	*Standard* Z
	newline
$\bullet$	$\bullet$
$\mid$	$\mid$
;	;
:	: ==
$\Leftrightarrow$	$\Leftrightarrow$
$\Rightarrow$	$\Rightarrow$
$\vee$	$\vee$
$\wedge$	$\wedge$
$\neg$	$\neg$
predicate pre (section 2.5.2)	
prefix and infix relations	relational predicates
if then else	if then else
	$\gg$
	$\fatsemi$
	$\setminus$
	$\upharpoonright$
	pre
infix generics	
$\times$	
infix functions	operator templates,
$\mathbb{P}$	with $\times$, $\mathbb{P}$, *etc*, at
prefix generics	same precedence
$(- \_)$	
$(\_ (\!\mid \_ \mid\!))$	
juxtaposed function application	juxtaposed function application
postfix functions	
	decoration
	renaming
selection	selection
θ	θ

Table 2.2 Operator precedences

Z, one of these schemas used as a predicate is equivalent to the corresponding predicate involving the operand schemas used as predicates. By using the same precedences, that ambiguity can be resolved arbitrarily. The remaining schema opera-

tors ($\gg$, $\fatsemi$, $\backslash$, $\upharpoonright$, pre) are given precedences adjacent to those of other expression-forming (functional) operators, and hence bind more tightly than they do in *ZRM*. The *ZRM* column omits some operators (namely *newline*, $==$, *decoration*, and *renaming*) because *ZRM* notation permits their use in only restricted contexts.

Type errors are likely as a result of unintended parses, but are not guaranteed.

2.4.8 Theta Expressions

Standard Z requires the types of components in the operand schema to be the same as the types of the same names in the current environment, unlike *ZRM*.

Mismatching types is likely to be indicative of a mistake, and, in those cases where it isn't, binding extensions provide an alternative notation.

A typechecker will detect and report all such problems.

2.4.9 Lexis of Words

The improvements in the lexis of Words (section 2.3.12) leads to a small incompatibilitiy. For example, λx is lexed as a single word.

ZRM views λ as a symbol and x as a letter, whereas *Standard* Z inherits Unicode's classification of both of them being letters.

Wherever two words are parsed as one, or one word is parsed as two, syntax or type errors are likely to result. White space should be inserted, e.g. $\lambda\ x$, or the single word renamed, as appropriate to conform to *Standard* Z.

2.5 Subtle Changes

This section discusses some subtle changes in the interpretation of certain notations that nevertheless leave the semantics of *ZRM* notation unchanged.

2.5.1 Quantified Expressions

Consider the schema quantification expression $\forall S \bullet T$. *ZRM* requires all names in S to be declared in T, too. *Standard* Z relaxes this requirement, for consistency with the scope rules of quantified predicates. For example, the schema expression $\forall x : \mathbb{A} \bullet T$, where $T == [\ y : \mathbb{A}\ ]$, is erroneous in *ZRM* (because x is not declared in T), but acceptable in *Standard* Z.

2.5.2 Preconditions

ZRM notation has pre predicates and pre schema expressions. *Standard* Z has only pre expressions. *ZRM* pre predicates are parsed as expressions, and those expressions are treated as schema predicates, giving a backwards-compatible effect.

2.5.3 Schema Instantiation

References to generic schemas no longer have to be given explicit instantiations, so long as the instantiations can be determined from the context. On the other hand, a reference to a generic schema in a theta expression is now permitted to have an explicit instantiation.

2.5.4 Precedence of lambda and mu

ZRM notation requires all λ and μ expressions to be parenthesized. *Standard* Z gives them precedences, so that parentheses can often be omitted. Parentheses are still required in the case of a μ expression whose $\bullet$ part is omitted.

2.6 Conclusions

There are several changes in *Standard* Z relative to *ZRM*. These are at the cost of some backwards incompatibilities. Although several pages have been devoted to the incompatibilities, they are all relatively minor compared to the improvements. It is hoped that the changes presented resolve satisfactorily the known inadequacies in the notation of *ZRM*.

There are further changes that could be made to Z. Some suggestions include: user-defined schema operators; operators for manipulating and removing decorations; further lexical issues (fonts, 2-dimensional templates for 'over-bar'-like symbols and matrices, etc). However, *Standard* Z goes a long way to removing many of the deficiencies that have become irksome as Z has increased in popularity and scope of use.

Chapter 3

Patterns Catalogue Conventions

These instances could be supplemented by many others, but they will serve to indicate how slow a process is the evolution of workable symbolism, and to point out that the process is far from complete.

— entry on 'Mathematical Notations',
Encyclopedia Britannica, 14th edition, 1949.

3.1 Background

The Z language is extensible. The ISO Standard includes a Standard Z Mathematical Toolkit (adapted from [Spivey 1992]), written in Z itself, which defines many of the sets, relations and functions used in Z specifications. A user is at liberty to add further new sets, relations and functions that may be used for a document, or for all of a specification containing many documents, or for all of the work of a particular author or at a particular installation, for example. (For example, see the Application-oriented theory pattern in [Stepney *et al.* 2003].)

This Patterns Catalogue volume presents a larger Toolkit, different parts of which we envisage would be useful to the majority of Z specifiers. It includes restatements, enhancements and modifications of the definitions in the Standard Z Mathematical Toolkit. The changes are of the following kinds:

1. alternative briefer definitions with identical effect
2. changes in the meaning of existing named sets, relations and functions
3. names for new sets, relations and functions

3.2 Generalisations

Standard Z introduces a new numeric type, whose most general form is given by the set $\mathbb{A}$, with subsets the integers, $\mathbb{Z}$, and the natural numbers, $\mathbb{N}$. In addition, we introduce further subsets: the real numbers, $\mathbb{R}$, and the rational numbers, $\mathbb{Q}$. We have the subsetting

$$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subseteq \mathbb{A}$$

where all of these sets are considered to be of the same type. We give axioms to characterise $\mathbb{R}$ and its subsets, leaving the possibly wider set $\mathbb{A}$ undescribed. Defining numbers in this way means that it is permissible for a Z specifier to widen the set of numbers while preserving all the properties we give here.

We have generalised most functions and relations that use numbers, and have introduced some new ones. All such modified relations and functions have their meaning extended over a wider domain than previously stated. This applies mainly to functions with numeric parameters, and takes three forms:

1. functions and relations involving integers have been generalised to allow real parameters wherever appropriate
2. functions and relations on finite sequences have been generalised to allow infinite sequences wherever appropriate
3. functions and relations involving any sort of number are stated in a way which defines their meaning for parameters in the intended domains, but leaves unstated the actual extent of the domain, so that a user may add to the meaning of the function or relation without falsifying the axioms of this Toolkit

The new versions here are supersets of the narrower versions in the Z Standard. This means that existing legal Z specifications remain legal except insofar as they explicitly refer to domains and ranges of standard functions and relations.

These generalisations cannot make any particular proof harder than it was before. This is because any properties that are true under the Standard Z Toolkit definition with the narrower domain remain theorems applicable to at least that domain, and once the old definitions are proved to be subsets of the new, a task that need be done once only, all proofs can be inherited. To work entirely within the narrower domains, one can carry on exactly as before.

3.3 Criteria for inclusion of definitions in the toolkit

When considering desirable toolkit content we need to remember the needs of several categories of user, such as specifiers, specification readers, tool-builders, and textbook writers. For each of these we can consider the impact of new toolkit content at successive levels of Z:

The lexical level. There is no problem when a toolkit gives meaning to ordinary alphabetic names, but wherever there is the introduction of new symbols, work is made for tool-builders. It also raises the familiarisation load for, in particular, readers. Learning the funny characters is off-putting to Z novices, and it seems that even experienced Z users are near the limit of what they want to remenber.

We introduce new characters and symbols only where they are already familiar in mathematics, (such as π) or, sparingly, where there is a real need. In particular, there does not seem to be a sufficient case for the provision of a large number of special characters for use with bags.

The syntax and type levels. There should not be a problem; our toolkit is written in Z, so tools can just use the definitions.

The semantics level. It is crucially important that all toolkit definitions are consistent, since otherwise all specifications using them are inconsistent too. Apart from that, provision of extra toolkit definitions makes no difference to specifications in which they are not used. Where they are used, having a definition in the toolkit rather than in the specification itself is an effective form of specification reuse. The development of laws about toolkit definitions should raise the effective power of proof about specifications.

The names in scope. If the toolkit gives meaning to a name, it is undesirable, even if legal, to use that name globally for anything else. Names chosen for toolkit definitions should either be well-established in mathematics, like *cos*, or sufficiently long and explicit that accidental reuse is unlikely, like *totalOrder*. Note that if a specification makes frequent use of a concept which is in the toolkit with a long name, it is trivial for the specifier to give it a short synonym.

The cognitive and credibility level. It is important not to burden the toolkit with anything that is going to be perceived as silly or meaningless or ill-designed. This judgement, however, can only be made by experienced Z people who have

taken the trouble to understand what a proposed new definition means. We should not exclude anything of value just because it is unfamiliar to novices.

Redundant modes of expression. There can never be a canonical way of using a notation as powerful as Z, so there is always scope for taste in the way specifications are written. Nevertheless we should not add any definitions whose only effect is to allow an alternative to existing forms, unless they also allow some specifications to be shorter or more readable, or proofs easier.

Let us also give some positive reasons for including things in the toolkit.

- A useful concept is well-established in mathematics, but the definitions in the literature differ in detail (for example, *totalOrder*). A standardised definition allows all Z users to speak the same language.
- A concept is well-established in mathematics, but the Z community, although showing a need for it, has not grown accustomed to expressing it in Z (for example: the road leading to the eventual specification of π in §33). With these well established concepts available in this toolkit, specifiers should find it easy to add further such definitions for themselves.
- A concept is well-established in mathematics, and can be used as a brief and elegant way of defining concepts that are to be included anyway (for example, monoids and groups, used in the definition of numbers).
- A concept closes up the algebra of some part of Z, or improves the semantic neatness of the system (for example, symmetric set difference).

3.4 Layout

Each of the toolkit chapters introduces several constructs, in the form of *patterns*. Each pattern has the following structure:

Name of the pattern

Intent
A short description of what the pattern is for.

Syntax/Definition
For the core language, the syntax of the construct. For the toolkit proper, the mathematical definition of the constuct. These definitions may make use

of earlier definitions, but there are no forward references.

Examples

A few simple examples to explain the syntax or definition of the construct, with a diagram if appropriate; a few examples of its typical use in a specification. These latter examples may include forward references to other constructs not yet defined: specification-before-use might be required for definitions, but it can lead to a very unmotivated style of description.

Laws

Some of the laws that the construct obeys, that may be used when reasoning about a specification. These laws can also help in understanding the properties of the construct. Again, laws sometimes make use of constructs not yet defined. Any examples of laws are typeset in a smaller font, to help distinguish them. Laws marked with ■ are proved in appendix D.

[Various]

Optional descriptions peculiar to the construct in question.

———————————•

3.5 Design

When designing a toolkit, one is often abstracting from existing specifications, trying to generalise the commonalities, and fill in obvious gaps. For example: we include symmetric set difference to close the set operator definitions; we have many associative operators, so this suggests including group; we have various distributed operators, so it suggests the general distributeOverSequence.

Many of our definitions are generic. When writing a generic definition, there are questions to bear in mind: does the axiomatic definition (of a relation or whatever) depend on the properties of *all* the types in the declaration? If not, can it sensibly be made generic in those types? If so, can it be sensibly generalised? Can it be made into a family of definitions? Is it actually (close to) an existing mathematical concept that can be reused? If there is a cluster of definitions, are there any more definitions that are needed to complete the cluster in some way?

When using a particular set in a declaration, $\mathbb{N}$ say, questions include: are *all* its properties needed (arithmetic, say) or just *some* of them (that it is ordered)? If not all the properties are being used, generalise it to the less constrained set that has just those properties needed.

(Future versions of the Pattern Catalogue will capture these guidelines more formally as process patterns.)

3.6 Presentation conventions

> *It was said of Jordan's writings that if he had 4 things on the same footing (as a, b, c, d) they would appear as a, M_3', ϵ_2, $\Pi''_{1,2}$.*
>
> — J. E. Littlewood, *Littlewood's Miscellany*, 1986

3.6.1 Specification style conventions

In Z, there are often many different ways to say the same thing. When we give examples, we try to exercise these differences, in order to show the available range. However, there are times when this could be confusing, so then we stick to a convention.

In particular, a naming convention is important for reuse, because it can make things easier to understand in the first place, and easier to refresh your memory when looking up a definition ([Barden *et al.* 1994, chapter 8], the Name consistently pattern in [Stepney *et al.* 2003].) Designers of (good) object oriented class libraries, for example, take great care over method naming across related classes.

3.6.2 Distinct elements

In some examples and laws, we wish to introduce several named elements, for example $x, y, z : X$, and we wish these to be distinct. There are several ways we could express this distinction, for example

- Say they are not equal: $x \neq y \neq z \neq x$
  This does not extend well to many items.
- Say they are disjoint: $\mathrm{disjoint}\langle\{x\}, \{y\}, \{z\}\rangle$
  This uses rather a lot of symbols for such a simple concept.
- Say how many of them there are: $\#\{x, y, z\} = 3$
  This is the style we use.

3.6.3 Diagramming conventions

A diagram is not a definition, just an example. We use conventions, described in appendix A, to make these examples more uniformly understandable.

There is a great danger with using a diagram, as with using any other example, that more will be read into it than is intended. The most obvious danger is that a diagram is finite, and hence 'finiteness' may be thought to be a distinguishing property. This is not so. Although on occasion there seems to be a vague assumption in some specifications that sets are naturally finite, with funny special cases being infinite, in reality, the opposite holds: *sets known to be finite are the special case.*

3.6.4 Global naming conventions

External names affect the meaning of a specification. They are often chosen to be quite long, so as not to clash unexpectedly with existing names. In this catalogue, textual global names (as opposed to symbolic ones) are written in a roman (non-italic) font. This helps to distinguish them visually (although not formally) from local names, and also, in large specifications, from user-introduced global names.

We also use a capitalisation convention to distinguish different kinds of definitions:

- given sets (including free types): ALL_CAPITALS
- schema names: UpperStartMixedCases
- all other names: lowerStartMixedCase or symbolic

Sometimes a concept being specified already has a well-known name that does not obey our capitalisation convention. In such a case we use the well-known name, but write it in a sans serif font, to indicate that our convention is not being used. For example

$$OPCODE ::= \mathsf{AND} \mid \mathsf{TAD} \mid \mathsf{ISZ} \mid \mathsf{DCA} \mid \mathsf{JMS} \mid \mathsf{JMP}$$

3.6.5 Local naming conventions

We use an italic font for local names. Subscripts are used to indicate a particular subset of the more general set.

- meta-parameters: for syntactic elements that cannot be quantified over or made generic in the core language: $\mathcal{E}$ for an arbitrary expression; $\mathcal{N}$ for an arbitrary name
- generic parameter: $[X, Y, Z]$ generally; $[L]$ for a labelling parameter; $[\dagger S, T]$ for type constrained generics
- given set values: $x, y, z : X;\ y : Y;\ l : L$

- set valued: $a, b, c : \mathbb{P}\, X$; $\alpha, \beta : \mathbb{P}(\mathbb{P}\, X)$
  non-empty sets: a_1; finite sets: a_f;
- pair valued: $p, q : X \times Y$
- relation valued: $r, s, t : X \leftrightarrow Y$; $\rho, \sigma : \mathbb{P}(X \leftrightarrow Y)$
- order valued: $\_ \lessdot \_ : \mathit{order}\ X$; $\_ \prec \_$: irreflexiveOrder X; $\_ \preccurlyeq \_$: reflexiveOrder X
- function valued: $f, g, h : X \nrightarrow Y$
  total functions: f_t; injections: f_i; surjection: f_s; bijection: f_b.
- sequence valued: $s, t, u, v : \mathrm{seq}\, X$
- bag valued: $b, c : \mathrm{bag}\, X$
- natural valued: $n, m : \mathbb{N}$
- prime valued: p : prime
- integer valued: $i, j, k : \mathbb{Z}$
- real valued: $x, y, z : \mathbb{R}$

Dashes are used for variables with a similar role. For example, we use x, y, y' if y and y' have a similar relationship to x, but would use x, y, z if they do not.

The name lists above overlap (for example, b could be a set valued or bag valued name), but, since a name is used only in the scope of a declaration, it is clear from such context which one is meant.

3.6.6 Consistency

> *A foolish consistency is the hobgoblin of little minds.*
>
> — Ralph Waldo Emerson, 'Self-Reliance', 1841

It is important to use conventions consistently, because this aids readability and understandability. However, it is also important to remember the reasons for these conventions. Where their use would in fact decrease readability, possibly because two conventions clash confusingly, they should not be followed slavishly. On those rare occasions in what follows, we abandon the conventions above, and use a more flexible approach.

3.7 Generic parameters

3.7.1 Implicit generic parameters

Many of the definitions we give are *generic*; on use the generic parameters can be instantiated with some particular set, or left implicit, in which case they default to the type, which is usually what is wanted. In only a few circumstances is it necessary to supply actual generic parameters, in order to provide sufficient type information. On rare occasions a piece of Z may be more understandable when generic parameters are given; for example, when an empty set appears twice, with different instantiations:

$$X = \varnothing \Leftrightarrow \mathbb{P}_1 X = \varnothing$$

On such occasions only, we give explicit instantiation:

$$X = \varnothing[X] \Leftrightarrow \mathbb{P}_1 X = \varnothing[\mathbb{P} X]$$

3.7.2 Generic schemas

In several places we define a compound mathematical structure using a schema to gather together the various components. For example, Homomorphism (§21) and Group (§23).

These structures are generic, and involve various sets and functions over those sets. For example, we choose to define a *Homomorphism* using the following generic schema:

*Homomorphism*0$[X, Y]$

$$a : \mathbb{P} X;\ \_ \diamond_x \_ : X \times X \pfun X$$
$$b : \mathbb{P} Y;\ \_ \diamond_y \_ : Y \times Y \pfun Y$$
$$h : X \pfun Y$$

$$a^{2\times} \lhd (\_ \diamond_x \_) \in a \times a \fun a$$
$$b^{2\times} \lhd (\_ \diamond_y \_) \in b \times b \fun b$$
$$a \lhd h \in a \fun b$$
$$\forall x, y : a \bullet h(x \diamond_x y) = h\ x \diamond_y h\ y$$

We use the generic parameters, here X and Y, to capture the types of the sets, and we also explicitly define sets of interest, here a and b, requiring them to be

subsets of the relevant generic sets. We then require the functions to be total only on the possibly smaller explicit sets a and b.

We can then state that some particular sets and functions form a *Homomorphism* as:

$$\exists\, Homomorphism0[\mathbb{A}, \mathbb{A}] \bullet a = \mathbb{R}_+ \land (\_ \diamond_x \_) = (\_ * \_) \\ \land\ b = \mathbb{R} \land (\_ \diamond_y \_) = (\_ + \_) \\ \land\ h = \ln$$

We could have chosen a different definition, that takes the generic parameters to be the sets of interest, and requires the functions to be total on the generic parameter sets. For example:

*Homomorphism*1$[X, Y]$

$$\_ \diamond_x \_ : X \times X \to X$$
$$\_ \diamond_y \_ : Y \times Y \to Y$$
$$h : X \to Y$$

$$\forall\, x, y : X \bullet h(x \diamond_x y) = h\ x \diamond_y h\ y$$

We would have then stated that some particular set and function form a *Homomorphism* as:

$$\exists\, Homomorphism1[\mathbb{R}_+, \mathbb{R}] \bullet (\_ \diamond_x \_) = (\mathbb{R}_+ \times \mathbb{R}_+) \lhd (\_ * \_) \\ \land\ (\_ \diamond_y \_) = (\mathbb{R} \times \mathbb{R}) \lhd (\_ + \_) \\ \land\ h = (\mathbb{R}_+ \times \mathbb{R}_+) \lhd \ln$$

Although our choice leads to a slightly longer definition, it results in simpler use: we can define various structures with functions that have been defined over wider sets without having to restrict them at the point of instantiation.

3.7.3 Generic conjectures

(In this catalogue, we use an extended conjecture syntax; see §5, Conjectures : variant for details.)

There are some subtleties in the definition and instantiation of generic conjectures. Consider a generic theorem such as

$$[X, Y]\ f : X \to Y \vdash \mathrm{dom}\, f = X$$

and consider trying to use it to prove the conjecture

$$f : \mathbb{N} \to \mathbb{N} \vdash \operatorname{dom} f = \mathbb{N}$$

Leaving the generic parameters of *dom* implicit it appears that we can do the proof. But once the parameters have been instantiated, we see that the theorem is actually

$$[X, Y]\ f : X \to Y \vdash \operatorname{dom}[X, Y] f = X$$

and the conjecture is actually

$$f : \mathbb{N} \to \mathbb{N} \vdash \operatorname{dom}[\mathbb{A}, \mathbb{A}] f = \mathbb{N}$$

and so the theorem is inapplicable. If instead we write the theorem as

$$[X_0, Y_0]\ X : \mathbb{P}\, X_0;\ Y : \mathbb{P}\, Y_0 \vdash \forall f : X \to Y \bullet \operatorname{dom} f = X$$

then on instantiation it becomes

$$[X_0, Y_0]\ X : \mathbb{P}\, X_0;\ Y : \mathbb{P}\, Y_0 \vdash \forall f : X \to Y \bullet \operatorname{dom}[X_0, Y_0] f = X$$

and it is now applicable.

We choose not to clutter our conjectures in this way. If such a facility is required, any of our conjectures, which are of the form

$$[X, \ldots, Z]\ \mathcal{S} \vdash \mathcal{P}$$

may be recast as

$$[X_0, \ldots, Z_0]\ X : \mathbb{P}\, X_0;\ \ldots;\ Z : \mathbb{P}\, Z_0 \vdash \forall \mathcal{S} \bullet \mathcal{P}$$

as required.

> *Do I contradict myself?*
> *Very well then I contradict myself,*
> *(I am large, I contain multitudes.)*
>
> — Walt Whitman, 'Song of Myself', part 51, 1855

3.8 Loose generics

As we mentioned in §2.3.8, *Standard* Z allows us to use *loose generic definitions*.

3.8.1 Extendable definitions

Our later definitions use $\mathbb{A}$, a set of numbers big enough to contain at least the reals, and with enough freedom to be extendable in different ways by different specifiers. When we define functions over $\mathbb{A}$s, we define them for no more than reals, leaving the definition loose outside this set.

Even if we restrict ourselves to just real numbers, not all functions are defined over all sets of reals. For example, the sum of a series (modelled as a labelled set of numbers) is defined only if the series is *convergent*. Our approach is to give a declaration wide enough to allow the operation on all labelled sets of $\mathbb{A}$s, but to *define* it only on a subset, leaving it loose outside this set, allowing different extensions as appropriate. So we declare such a sum operator as:

$$\begin{array}{|l}
\hline
[L] \\
\Sigma \ : (L \pfun \mathbb{A}) \pfun \mathbb{A} \\
\hline
\ldots \\
\hline
\end{array}$$

We have a style for defining functions in an extendable way. We define a total function (often by using a λ-expression) over the subdomain of interest, and say that it is a subset of the wider, loosely defined function.

$$\begin{array}{|l}
\hline
[\,Y\,] \\
op : \mathbb{A} \pfun Y \\
\hline
(\ \lambda\, x : \mathbb{R} \bullet \mathcal{E}_{xy}\ ) \subseteq op \\
\hline
\end{array}$$

(Here $\mathcal{E}_x$ represents some expression involving x and y.) This allows some later extension of the definition to a wider subdomain, but the definition within the subdomain cannot be changed. Any such change must either remove a pair from the relation, which would violate the subset condition, or add a pair to the relation, which would violate functionality. (It cannot change a pair, becuase that would not be an *extension*.) Hence a consistent change of the definition inside the subdomain is not possible.

We also have a style for defining relations in an extendable way. Over the subdomain of interest we uniquely define the relation; outside the subdomain we leave the definition loose. This allows some later extension of the relation to a wider subdomain, but the definition within the subdomain cannot be changed.

3.8.2 Example: loose choice function

The ability to define loose generics allows a simple form of operator overloading and polymorphic definitions. An example of using a loose generic definition in this way is the following choice function (suggested by Rob Arthan). It is loose in the sense that it avoids specifying which component of the given pair is returned by the function.

$$[X]$$
$$pickFromPair : X \times X \to X$$
$$\forall x, y : X \times X \bullet pickFromPair(x, y) \in \{x, y\}$$

In combination with global constraints (which can be generic), loose generics provide an *ad hoc* overloading mechanism without any additional notation. For example, different instantiations of the generic definition *pickFromPair* can be constrained to behave differently.

$$pickFromPair[\mathbb{N}] = first$$
$$pickFromPair[\mathbb{R}] = second$$

This paragraph outline reflects *Standard* Z's view of a global constraint as being an axiomatic paragraph with no declaration.

3.8.3 Example: polymorphic addition operator

Another example of using a loose generic definition to provide polymorphism is the following addition operator (suggested by Jim Woodcock).

We can declare a generic operator

$$[X, Y]$$
$$op : X \pfun Y$$

and then extend its definition to cover the required types of arguments

$$\forall a : A \bullet op\ a = \mathcal{E}_{ab}$$
$$\forall n : N \bullet op\ n = \mathcal{E}_{nm}$$

Hence the same operator name can be used for different types, with different patterns of behaviour. The generic declaration requires it to be functional.

We later declare our arithmetic operators over the set of numbers $\mathbb{A}$. We could instead choose to define them generically, allowing them to be extended later for use with a different type of argument, as

$$[X]$$
$$\_ + \_ : X \times X \pfun X$$

and then numeric addition, matrix addition, and sequence concatenation (say) could all use the same symbol.

$$\forall x, y : \mathbb{R} \bullet x + y = \text{(definition of arithmetic addition)}$$
$$\forall m, n : \mathcal{M} \mid \mathit{compatible}(m, n) \bullet m + n = \text{(definition of matrix addition)}$$

$$[X]$$
$$\forall s, t : \seq X \bullet s + t = \text{(definition of sequence concatenation)}$$

Style point: Although there is now the ability to write loose generics, the overloading that they enable has been little used in practice as yet. Such 'operator overloading' should be used only for the same 'kind' of operation. There is potential for obfuscation, so use loose generics with care.

3.9 Operator template paragraphs

We exploit the introduction of operator template paragraphs in our style of specification. We usually specify relations and functions explictly, rather than using quantifiers. For example, we specify the relation 'not equal to' with

$$\text{relation } (\_ \neq \_)$$
$$\_ \neq \_ [X] == \{ x, y : X \mid \neg\, x = y \}$$

rather than as

$$[X]$$
$$\_ \neq \_ : X \leftrightarrow X$$
$$\forall x, y : X \bullet x \neq y \Leftrightarrow \neg\, x = y$$

and the function 'set difference' as

function 30 leftassoc (_ \ _)
$\_ \setminus \_\ [X] == \lambda\, a, b : \mathbb{P}\, X \bullet \{\ x : X \mid x \in a \land x \notin b\ \}$

rather than as

$$[X]$$
$$\_ \setminus \_ : \mathbb{P}\, X \times \mathbb{P}\, X \to \mathbb{P}\, X$$
$$\forall a, b : \mathbb{P}\, X \bullet a \setminus b = \{\ x : X \mid x \in a \land x \notin b\ \}$$

We are able to use these new forms because the operator template paragraphs clearly distinguish between infix functions and infix relations. We feel that our style of definition is superior to the quantified style because it is shorter; because the proof that it defines something is immediate (whereas the quantifed style often requires a less trivial proof that the object fulfilling the stated conditions actually exists); and because the quantified form is more easily derived from the explicit form than *vice versa*. We often provide the equivalent quantified form as a law.

Part II

Core Language

Chapter 4

Core meta-language definitions

4.1 Introduction

Some constructs are part of the core Z language (that is, not defined in Z themselves as part of a toolkit). Their syntax, type and meaning needs to be defined separately, using some meta-language.

In this part we define the syntax of the core language constructs, and describe their type and meaning where appropriate. (Future versions of this catalogue will define the type rules and meaning functions, as done in the ISO Z Standard document [ISO-Z 2002], and will include explanatory and tutorial material.)

It is useful to have laws about such constructs, and some are given in this part.

4.2 Syntax notation

The notation we use to describe *Standard* Z syntax is as follows. (The meta-symbols used in the descriptions are given in `typewriter` font, to distinguish them from similar-appearing Z symbols.)

- **sequence:** juxtaposition: $a\ b\ c$
- **alternatives:** separated by vertical bars: a `|` b `|` c
- **optional:** enclosed in square brackets: `[` a `]`
- **zero or more:** enclosed in braces `{` a `}`

Chapter 5

Specification, Section, Paragraph

Specification, Section

Intent

Provide a simple structuring mechanism, to permit reusable 'toolkit' sections to be added to specifications.

Syntax

```
Specification ::= { Section }
```

A *Standard* Z *Specification* comprises a list of *Section*s.

A *Section* may have a header, which gives it a name, and if so may include one or more parent sections (a *Section* without a header is 'anonymous'). The body of a section is a list of *Paragraph*s.

```
Section ::= [ section NAME [ parents NAME { , NAME } ] ]
            { Paragraph }
```

Meaning

For a *Specification* to be well-formed, each of a section's parents must occur as a previous section.

For backwards compatibility, an anonymous section (a single sequence of paragraphs with no section header) is accepted as a single section with *toolkit* (the Standard Z Mathematical Toolkit) as its sole parent. An anonymous section cannot be the parent of any other section.

The meaning of the whole specification is derived from the last listed section and the inclusion of each of its parents, their parents, and so on. The Standard Z Prelude (§29) is a default parent of every section.

Sections provide *Standard* Z with a minimal form of modularisation. It is *not* an encapsulation mechanism: a section does not hide any of its paragraphs. So if a section is included as a parent, *all* its global declarations become visible; there is nothing equivalent to an 'export' clause.

Global redeclaration is not permitted, either within a section or across related sections. A name may be included by different routes through the parent hierarchy provided that it was originally declared in a single parent section.

The *Standard* Z toolkit, and the declarations we provide here, are composed into one section per chapter.

———————————————•

Paragraph

Syntax

```
Paragraph ::= GivenSet
            | FreeType
            | AxiomaticDeclaration
            | HorizontalDeclaration
            | SchemaBoxDeclaration
            | GenericAxiomaticDeclaration
            | GenericHorizontalDeclaration
            | GenericOperatorDeclaration
            | GenericSchemaBoxDeclaration
            | Conjecture
            | GenericConjecture
            | OperatorTemplate
```

There are three kinds of *Paragraph*:

1. Global declarations, which introduce new names, with their types and values, or further constrain previously declared names. There are three kinds of names that can be introduced:

(a) given sets, either unconstrained, or constrained as free types (covered in §9)
(b) simple declarations, including schemas as a special case
(c) generic declarations, including generic schemas as a special case

2. Conjectures, both simple and generic, which assert properties of the specification.
3. Operator templates, which define the 'shape', precedence and associativity of mixfix operators.

Simple declarations

Intent

There is one main form of simple declaration, the *axiomatic declaration*, with two special cases, the *horizontal declaration* (capturing both of *ZRM*'s 'abbreviation definition' and 'horizontal schema declaration'), and the *schema box declaration*.

Syntax

```
AxiomaticDeclaration ::= AX SchemaText

HorizontalDeclaration ::= ZED EqualityDeclaration
EqualityDeclaration   ::= DeclName == Expression

SchemaBoxDeclaration ::= SCH NAME SchemaText
```

Description

Standard Z's *axiomatic declaration* is similar in concept to *ZRM*'s axiomatic declaration, with the main changes being due to liberalisation of the syntax elsewhere:

- The declaration part may include 'equality declarations' as well as 'colon declarations'
- The declaration part may include schema expressions, as well as 'schema references'

- The declaration part may be empty

The *horizontal declaration* can be used where a single name is being given a single value (which may itself be set valued). A horizontal declaration

$$\mathcal{N} == \mathcal{E}$$

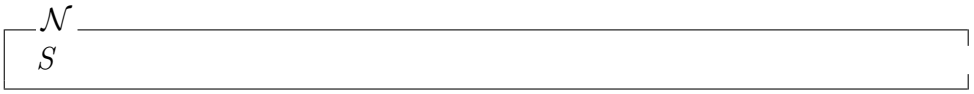
is equivalent to the axiomatic declaration

$$\mid \quad \mathcal{N} == \mathcal{E}$$

which is also equivalent to

$$\mid \quad \mathcal{N} : \{\mathcal{E}\}$$

The schema box declaration



is equivalent to the horizontal declaration

$$\mathcal{N} == [\ S\ ]$$

The vertical box form tends to be chosen for use where a schema definition is large, in order to emphasise the extent of the formal text, and the horizontal form when it is small, in order to save space.

Generic declarations

Intent

There are generic forms of each of the simple declaration paragraphs, allowing the provision of generic parameters. The *generic horizontal declaration* form is split into two kinds, one for simple names, one for mixfix operator names with the generic parameters placed in their slots.

Syntax

```
GenericAxiomaticDeclaration  ::= GENAX [ Formals ] SchemaText
GenericHorizontalDeclaration ::= NAME [ Formals ] == Expression
GenericOperatorDeclaration   ::= GenName == Expression
GenericSchemaBoxDeclaration  ::= GENSCH NAME [ Formals ] SchemaText
```

Description

The paragraph

$$X\ x_1\ \ldots\ x_n\ Z == \mathcal{E}$$

is equivalent to

$$[X, \ldots, Z]$$
$$\_\ x_1\ \ldots\ x_n\ \_ == \mathcal{E}$$

(and similarly for other styles of templates).

Variant

We use *schema-constrained generics* [Valentine *et al.* 2000] in some of our laws. These are generic parameters whose types are constrained to be schemas, allowing us to express laws about general schemas in a natural manner.

The constrained parameters are written after a dagger in the generic parameter list. So $[X, Y \dagger S, T]$ indicates the usual unconstrained generic parameters X and Y, and the variant schema-constrained parameters S and T. (Further type constraints may be implicit in the use of S and T, for example, they might need to have compatible signatures. Such constraints are inferrable. See [Valentine *et al.* 2000] for details.)

•

Conjectures

Intent

Express desired properties of a specification.

Syntax

`Conjecture ::=` $\vdash?$ `Predicate`

`GenericConjecture ::=` [`Formals`] $\vdash?$ `Predicate`

Meaning

A conjecture is *valid* if its predicate can be shown to be implied by the properties of the specification, without itself contributing to those properties.

A conjecture is not required to be valid in a well-formed specification; it must merely be well-typed. It is said to be a *theorem* if and only if it is valid.

Motivation

If one writes a predicate paragraph

$$\vdash \quad Predicate$$

then the meaning of this paragraph is to restrict the specification to environments where the *Predicate* is *true*. For example, we could introduce a number n

$$n : \mathbb{N}$$

and then the predicate that n is *even*

$$\vdash \quad n \in even$$

The effect of the predicate paragraph is to *constrain* n to be *even*.

We can have a paragraph that states the *Predicate* is *true* for *all* environments. For example, there is always a number greater than n:

$$\vdash?\ \exists\, m : \mathbb{N} \bullet n < m$$

This conjecture does not constrain n to those values that have such a property. Rather, it states that it should be true for all n (as constrained by the other paragraphs) — it may require a proof to establish this fact. So a conjecture adds no new constraints to the specification.

Variant

In our statements of laws later, we claim these are theorems (that they are *true* in all environments) by dropping the conjecture question mark.

We also separate out declarations from the 'body' of the theorem. We use

$$D \mid P \vdash Q$$

as syntactic sugar for

$$\vdash \forall D \mid P \bullet Q$$

and

$$[X]\ D \mid P \vdash Q$$

as syntactic sugar for

$$[X] \vdash \forall D \mid P \bullet Q$$

Examples

1. $a : \mathbb{Z} \vdash a \mathinner{..} a = \{a\}$
2. $[X]\ \alpha : \mathbb{P}_1 \mathbb{P} X;\ a : \mathbb{P} X \vdash a \setminus \bigcup \alpha = \bigcap \{b : \alpha \bullet a \setminus b\}$
3. an invalid, but still well-formed, conjecture: $\vdash 42 \in \{1, 2, 3\}$

---•

Operator templates

Intent

Define general *mixfix* symbols, with specified associativity and precedence.

Syntax

```
OperatorTemplate ::= Category [ Assoc ] ( Template )

Category ::= relation | function | generic

Assoc ::= NUMBER ( leftassoc | rightassoc )

Template ::= [ _ ] NAME [ { Slot NAME } ] [ _ ]
Slot     ::= _ | ,,
```

There must be at least one slot in the template.

Motivation

The operator template paragraph merges the separate *ZRM* categories of prefix and infix relations (like disjoint $\_$ and $\_ < \_$), left-associative infix and postfix operators (like $\_ + \_$ and $\_^*$), and generic prefix and right-associative infix operators (like $\mathbb{F}\ \_$ and $\_ \rightarrow \_$) into a single uniform mechanism.

It is not necessary to hardwire sequence brackets, relational image symbols, and the right associative unary minus, into the syntax. They can each be defined using a template in an operator template paragraph.

ZRM has a fixed table to define its operator precedence, that multiplication binds stronger than addition, say. The *Standard* Z operator template paragraph provides an extensible way to define precedence.

Table 5.1 summarises what is permitted. The precedences of the *Standard* Z toolkit operators have the same order as in *ZRM*. *ZRM* is however more restrictive than suggested by the table: relations and generics cannot be postfix, functions cannot be declared to be prefix (they just are by default), and there are no nofix (bracketting) operators.

	ZRM	*Standard* Z
category	relation, function, generic	relation, function, generic
precedence	infix functions 1..6, others fixed by syntax	infix functions and generics $a..b$, others fixed by syntax
associativity	left or right, fixed by syntax	left or right, user-defined
arity	1..2	1..n, operands and symbols alternate
sequence arguments	no	yes

Table 5.1 Operators

Category

There are three categories of fixity paragraph, distinguished by the keyword *relation*, *function* or *generic*.

An operator's category determines how applications of the operator are parsed: an application of a relation operator is parsed as a relational predicate; an application of a function operator is parsed as a function application expression; an application of a generic operator is parsed as a generic instantiation expression.

Function and relation operators that are generic usually have their generic arguments left implicit. If for some reason those arguments are explicit then they appear in square brackets, in either *ZRM* or *Standard* Z.

Precedence and Associativity

The *Assoc* part of the template specifies precedence and associativity for the new operator. It is required only for function and generic infix templates.

When applications of operators are nested, so that one operator application appears as an operand in another operator application, the intended nesting can be made explicit using parentheses. Alternatively, if no parentheses are used, the precedence and associativity information determines how the applications are nested: applications of operators with higher precedence bind more tightly than ones of lower precedence; nested applications of infix operators with the same precedence associate either to the left or right as declared. All operators sharing the same precedence must have the same associativity.

A left associative unary operator is simply a function application. An example of a right associative unary operator is unary minus. Because it is right associative, the expression $--x$ means $-(-x)$, as expected; if it were left associative it would mean $(--)x$, not what is wanted.

The operators in the *Standard Z* toolkit have precedences chosen to reproduce those of *ZRM* as far as possible. Here, we take an infix generic (such as $\rightarrow$) to bind less tightly than Cartesian product $\times$. So $A \times B \rightarrow C \times D$ means $(A \times B) \rightarrow (C \times D)$. This is equivalent to Cartesian product having a binding strength between 5 (that of $\rightarrow$) and 10 (that of the weakest binding infix functions).

Avoid making heavy use of precedences: although most readers are happy for negation to bind more tightly than multiplication, which binds more tightly than addition, more levels than this probably become confusing. Choose your own operators to have one of the 'standard' precedences (for example, infix generics at 5), and use brackets as necessary.

Fewer brackets are needed with *Standard* Z than with *ZRM*. For example, in *Standard* Z $\mathbb{P}\,\mathbb{P}\,\mathbb{P}\,X$ means $\mathbb{P}(\mathbb{P}(\mathbb{P}\,X))$; $\mathrm{seq}\,\mathrm{seq}\,\mathrm{seq}\,a$ means $\mathrm{seq}(\mathrm{seq}(\mathrm{seq}\,a))$; $\mathbb{F}\,\mathrm{seq}\,\mathbb{P}_1\,a$ means $\mathbb{F}(\mathrm{seq}(\mathbb{P}_1\,a))$. In *ZRM* the unbracketed forms are all syntax errors.

Templates

The *Template* defines the 'shape' of the mixfix symbol. It is made of alternating argument slots and words (parts of the symbol). For example, the relational image template is $\_(\!|\ \_\ |\!)$.

An argument slot can be a normal argument (_), or a list argument(, ,). When a template is used in a specification, each normal argument slot is filled with a single expression, for example a template like $\_ \diamond \_$ can be used as $n \diamond \# b$, and each sequence argument slot is filled with a comma separated list of expressions, for example a template like $\{|, , |\}$ can be used as $\{\!|\ n, \# b, 3, 42\ |\!\}$. When the meaning of the mixfix symbol is being defined, this comma separated list is treated as a *sequence* of expressions. (See §34 and §39 for examples of this kind of template in use.)

The use of arbitrary arity is subject to the restriction that operands slots and symbol names must alternate, meaning that two operands cannot be consecutive without an intervening symbol, and two symbols cannot be consecutive without an intervening operand. An example of the latter restriction would be the consecutive symbols *else if* (within the obvious operator), whereas writing them as a single symbol *elseif* would be permitted.

Templates may begin or end either with an argument slot or with a word, giving four possibilites:

- *infix*: begin and end with a slot; for example $\_ + \_$
- *prefix*: begin with a word, end with a slot; for example $\mathit{disjoint}\_$
- *postfix*: begin with a slot, end with a word; for example $\_(\!|\ \_\ |\!)$
- *nofix*: begin and end with a word; for example $\langle , , \rangle$

In *Standard* Z an operator name may be the name of an element or injection of a free type, may be the name of a let definition, and may be selected from a binding.

Chaining

Chaining of relations in *Standard* Z is exactly as permitted by *ZRM*. Only infix binary relations may be chained; so a chain may not commence with a prefix relation, nor end with a postfix relation, nor can tertiary or higher relations appear in a chain. (Technically, an approach could be developed in which these would all be possible, but they would never be good stylistically, so are not supported in this approach.)

Examples

1. infix relation, 'not equal': relation $(\_ \neq \_)$
2. infix function, 'union': function 30 leftassoc $(\_ \cup \_)$
3. postfix function, 'relational image': function $(\_(\!|\ \_\ |\!))$
4. postfix function, 'relational iteration': function $(\_\,\mathsf{up}\,\_\,\mathsf{down})$, typeset $\_^{-}$
5. nofix function, 'sequence brackets': function $(\langle , , \rangle)$
6. nofix function, *ZRM* 'bag brackets': function $([\![, ,]\!])$
7. prefix generic, 'finite sequence': function $(\mathrm{seq}\,\_)$
8. infix generic, 'relation': generic 5 rightassoc $(\_ \rightarrow \_)$
9. in a free type: $\mathit{TREE} ::= \mathit{node} \mid \_\mathit{foo}\_\langle\!\langle \mathit{TREE} \times \mathit{TREE} \rangle\!\rangle$
10. in a let definition: let $\_\mathit{foo}\_ == \lambda\, x, y : X \bullet x \diamond y$
11. selected from a binding: $(\mu\, S).\_\mathit{foo}\_$ where $S == [\ \_\mathit{foo}\_ : \ldots\ ]$
12. chaining: $a = b < c$; $x = y \in a \subseteq b$

Defining meanings

An operator template paragraph merely defines a new *name*: it does not define a meaning for that name. The meaning has to be defined in the usual way, usually in an axiomatic or generic definition of the name, or of decorated versions of the name. (See the subsequent catalogue for copious examples.)

•

Chapter 6

Predicates

Students may also bring an owl OR a cat OR a toad.
— J. K Rowling, chapter 5, *Harry Potter and the Philosopher's Stone*, 1997

A Z predicate either is true or is false (though sometimes we cannot say which). That is, the logic of Z is two-valued. There are other possible treatments of predicates, but they are not generally accepted in the Z literature nor recommended in *Standard* Z. Specifications written on the assumption of the usual two-valued logic are consistent with all alternatives that have been seriously proposed, and we keep to that usual logic.

`Predicate`	
`::= Predicate ( NL | ; ) Predicate`	low-precedence conjunction
`| ∀ SchemaText • Predicate`	universal quantification
`| ∃ SchemaText • Predicate`	existential quantification
`| ∃₁ SchemaText • Predicate`	unique existential quantification
`| Predicate ⇔ Predicate`	equivalence
`| Predicate ⇒ Predicate`	implication
`| Predicate ∨ Predicate`	disjunction
`| Predicate ∧ Predicate`	conjunction
`| ¬ Predicate`	negation
`| Relation`	relation operator application
`| Expression`	schema predicate
| true	truth
| false	falsity
| (`Predicate`)	bracketed predicate

Operator precedence increases down the list: $\wedge$ binds more tightly than does $\vee$, which binds more tightly than does $\Rightarrow$, which binds more tightly than does $\Leftrightarrow$.

The laws given here are 'generic' in schema types. We write S, T for a general

schema used as a declaration, and P, Q, R for a general schema used as a predicate. Two predicates are equivalent (have the same truth values) when $P \Leftrightarrow Q$.

Most of these laws are elementary properties of predicates, discussed in any book on logic; we do not prove them.

truth, falsity, and negation

Syntax

```
Predicate ::= ¬ Predicate
            | true
            | false
```

Laws

Law 6.1 The law of the excluded middle: a predicate is true precisely when it is not false.

$\vdash \mathit{true} \Leftrightarrow \neg\, \mathit{false}$
$[\dagger P] \ \vdash P \Leftrightarrow \neg \neg P$
$[\dagger P] \ \vdash P \vee \neg P$

•

Conjunction and disjunction

Syntax

```
Predicate ::= Predicate ( NL | ; ) Predicate
            | Predicate ∨ Predicate
            | Predicate ∧ Predicate
```

Laws

Law 6.2 The newline and semicolon predicates are low-precedence conjunctions, allowing conjuncts with operators of intervening precedences to be written with fewer parentheses.

$$[\dagger P, Q] \vdash (P\ Q) \Leftrightarrow (P) \wedge (Q)$$

$$[\dagger P, Q] \vdash (P;\ Q) \Leftrightarrow (P) \wedge (Q)$$

Law 6.3 Conjunction and disjunction are associative.

$$[\dagger P, Q, R] \vdash (P \wedge Q) \wedge R \Leftrightarrow P \wedge (Q \wedge R)$$
$$[\dagger P, Q, R] \vdash (P \vee Q) \vee R \Leftrightarrow P \vee (Q \vee R)$$

Law 6.4 Conjunction and disjunction are commutative.

$$[\dagger P, Q] \vdash P \wedge Q \Leftrightarrow Q \wedge P$$
$$[\dagger P, Q] \vdash P \vee Q \Leftrightarrow Q \vee P$$

Law 6.5 Absorbing *true* and *false*

$$[\dagger P] \vdash P \wedge \mathit{false} \Leftrightarrow \mathit{false}$$
$$[\dagger P] \vdash P \wedge \mathit{true} \Leftrightarrow P$$
$$[\dagger P] \vdash P \vee \mathit{false} \Leftrightarrow P$$
$$[\dagger P] \vdash P \vee \mathit{true} \Leftrightarrow \mathit{true}$$

Law 6.6 Conjunction and disjunction are idempotent.

$$[\dagger P] \vdash P \wedge P \Leftrightarrow P$$
$$[\dagger P] \vdash P \vee P \Leftrightarrow P$$

Law 6.7 de Morgan's laws (named for the British logician Augustus de Morgan, 1806–1871): negation pseudo-distributes through conjunction and disjunction.

$$[\dagger P, Q] \vdash \neg\,(P \wedge Q) \Leftrightarrow \neg\, P \vee \neg\, Q$$
$$[\dagger P, Q] \vdash \neg\,(P \vee Q) \Leftrightarrow \neg\, P \wedge \neg\, Q$$

Law 6.8 Conjunction and disjunction each distributes through the other.

$$[\dagger P, Q, Q'] \vdash P \wedge (Q \vee Q') \Leftrightarrow P \wedge Q \vee P \wedge Q'$$
$$[\dagger P, Q, Q'] \vdash P \vee Q \wedge Q' \Leftrightarrow (P \vee Q) \wedge (P \vee Q')$$

———————————————•

Implication

Syntax

```
Predicate ::= Predicate ⇒ Predicate
```

Laws

Law 6.9 Implication can be defined in terms of disjunction and negation.

$$[\dagger P, Q] \ \vdash P \Rightarrow Q \Leftrightarrow \neg\, P \lor Q$$

•

Equivalence

Syntax

```
Predicate ::= Predicate ⇔ Predicate
```

Laws

Law 6.10 Equivalence can be defined in terms of implication and conjunction. Equivalence is implication in both directions; equivalence means the predicates are either both true or both false.

$$[\dagger P, Q] \ \vdash (P \Leftrightarrow Q) \Leftrightarrow (P \Rightarrow Q) \land (Q \Rightarrow P)$$
$$[\dagger P, Q] \ \vdash (P \Leftrightarrow Q) \Leftrightarrow P \land Q \lor \neg\, P \land \neg\, Q$$

•

Universal quantification

Syntax

```
Predicate ::= ∀ SchemaText • Predicate
```

Laws

Law 6.11 If the predicate part is *true*, the entire universal quantification is *true*. That is, 'for every S, *true*' is equivalent to *true*.

$$[\dagger S] \vdash (\ \forall S \bullet true\ ) \Leftrightarrow true$$

Law 6.12 A predicate can be moved between the declaration part and the predicate part of a universal quantifer. That is, 'for every S constrained to satisfy P, it also satisfies Q' is equivalent to 'for every S, if it satisfies P then it also satisfies Q'.

$$[\dagger S, P, Q] \vdash (\ \forall S \mid P \bullet Q\ ) \Leftrightarrow (\ \forall S \bullet P \Rightarrow Q\ )$$

Law 6.13 Multiple declarations are equivalent to multiple quantifications.

$$[\dagger S_1, S_2, P] \vdash (\ \forall S_1;\ S_2 \bullet P\ ) \Leftrightarrow (\ \forall S_1 \bullet \forall S_2 \bullet P\ ) \qquad \text{[no variable capture]}$$

■ **Law 6.14** Universal quantification over the empty set is always true; universal quantification over a non-empty set is equivalent to conjunction.

$$[X \dagger P] \vdash (\ \forall x : \varnothing[X] \bullet P\ ) \Leftrightarrow true$$
$$[X \dagger P]\ y : X;\ a : \mathbb{P}\, X \vdash$$
$$(\ \forall x : \{y\} \cup a \bullet P\ ) \Leftrightarrow (\ \exists x == y \bullet P\ ) \wedge (\ \forall x : a \bullet P\ )$$

So any law about universal quantification implies a corresponding law about conjunction, by quantifying over a two-element set. Also, any law about conjunction of similar conjuncts can be generalised to a law about universal quantification over a *non-empty finite* set of such conjuncts (proof by induction on the size of the set). The case including quantification over the empty set, or over infinite sets, is not true in general.

Law 6.15 Universal quantification distributes through conjunction.

$$[\dagger S, P, Q] \vdash (\ \forall S \bullet P \land Q\ ) \Leftrightarrow (\ \forall S \bullet P\ ) \land (\ \forall S \bullet Q\ )$$

Law 6.16 A disjunct that does not depend on the quantified variable can be moved outside a universal quantification.

$$[\dagger S, P, Q] \vdash (\ \forall S \bullet P \lor Q\ ) \Leftrightarrow P \lor (\ \forall S \bullet Q\ ) \qquad [P \text{ does not depend on } S]$$
$$[\dagger S, P] \vdash (\ \forall S \bullet P\ ) \Leftrightarrow P \lor (\ \forall S \bullet \mathit{false}\ )$$

•

Existential quantification

Syntax

```
Predicate ::= ∃ SchemaText • Predicate
```

Examples

1. *ZRM*'s let predicate can be rewritten as an existential quantifier:

$$\begin{array}{ll} \quad \text{let } x_1 == e_1;\ \ldots;\ x_n == e_n \bullet P & \text{– } \textit{ZRM} \text{ only} \\ \equiv \exists\, x_1 == e_1;\ \ldots;\ x_n == e_n \bullet P & \text{– } \textit{Standard} \text{ Z only} \\ \equiv \exists\, x_1 : \{e_1\};\ \ldots;\ x_n : \{e_n\} \bullet P & \end{array}$$

   ZRM's let predicate conflicts with the let expression when P is a schema, and so is omitted from *Standard* Z.

Laws

Law 6.17 de Morgan's laws for quantifiers: negation pseudo-distributes through universal and existential quantification. (This law can be used to define existential quantification in terms of universal quantification, or vice versa.)

$$[\dagger S, P] \vdash \neg\, (\ \forall S \bullet P\ ) \Leftrightarrow (\ \exists S \bullet \neg\, P\ )$$
$$[\dagger S, P] \vdash \neg\, (\ \exists S \bullet P\ ) \Leftrightarrow (\ \forall S \bullet \neg\, P\ )$$

Law 6.18 If the predicate part is *false*, the entire existential quantification is *false*. That is, 'there is an S such that *false*' is equivalent to *false*.

$$[\dagger S] \vdash (\ \exists S \bullet \mathit{false}\ ) \Leftrightarrow \mathit{false}$$

Law 6.19 A predicate can be moved between the declaration part and the predicate part of an existential quantifer. That is, 'there is an S constrained to satisfy P, that also satisfies Q' is equivalent to: 'there is an S, that satisfies P and also satisfies Q'.

$$[\dagger S, P, Q] \vdash (\ \exists S \mid P \bullet Q\ ) \Leftrightarrow (\ \exists S \bullet P \land Q\ )$$

Law 6.20 Multiple declarations are equivalent to multiple quantifications.

$$[\dagger S_1, S_2, P] \vdash (\ \exists S_1;\ S_2 \bullet P\ ) \Leftrightarrow (\ \exists S_1 \bullet \exists S_2 \bullet P\ ) \qquad \text{[no variable capture]}$$

Law 6.21 Existential quantification over the empty set is always false; existential quantification over a non-empty set is equivalent to disjunction.

$$[X \dagger P] \vdash (\ \exists x : \varnothing[X] \bullet P\ ) \Leftrightarrow \mathit{false}$$
$$[X \dagger P]\ y : X;\ a : \mathbb{P}\, X \vdash$$
$$(\ \exists x : \{y\} \cup a \bullet P\ ) \Leftrightarrow (\ \exists x == y \bullet P\ ) \lor (\ \exists x : a \bullet P\ )$$

So any law about existential quantification implies a corresponding law about disjunction, by quantifying over a two-element set. Any law about disjunction of similar disjuncts can be generalised to a law about existential quantification over a *non-empty finite* set of such disjuncts (proof by induction on the size of the set). The case including quantification over the empty set, or over infinite sets, is not true in general.

Law 6.22 Existential quantification distributes through disjunction.

$$[\dagger S, P, Q] \vdash (\ \exists S \bullet P \lor Q\ ) \Leftrightarrow (\ \exists S \bullet P\ ) \lor (\ \exists S \bullet Q\ )$$

Law 6.23 A conjunction that does not depend on the quantified variable can be moved outside an existential quantification.

$$[\dagger S, P, Q] \vdash (\ \exists S \bullet P \land Q\ ) \Leftrightarrow P \land (\ \exists S \bullet Q\ ) \qquad [P \text{ does not depend on } S]$$
$$[\dagger S, P] \vdash (\ \exists S \bullet P\ ) \Leftrightarrow P \land (\ \exists S \bullet \mathit{true}\ )$$

———————————————•

Unique existential quantification

Syntax

$$\texttt{Predicate ::= } \exists_1 \texttt{ SchemaText } \bullet \texttt{ Predicate}$$

Laws

Law 6.24 A predicate can be moved between the declaration part and the predicate part of a unique existential quantifer. 'There is a unique S constrained to satisfy P, that also satisfies Q' is equivalent to: 'There is a unique S, that satisfies P and also satisfies Q'.

$$[\dagger S, P, Q] \vdash (\ \exists_1 S \mid P \bullet Q\ ) \Leftrightarrow (\ \exists_1 S \bullet P \land Q\ )$$

Law 6.25 Unique existence requires both the existence of some value that satisfies the predicate, and that no other distinct value also satisfies the predicate.

$$[\dagger S, P] \vdash (\ \exists_1 S \bullet P\ ) \Leftrightarrow (\ \exists S \bullet P \land (\ \forall [\ S \mid P\ ]' \bullet \theta S = \theta S'\ )\ )$$

•

Schema predicate

Intent

A schema expression (§8) can be used as a predicate.

A schema expression denotes a set of bindings, where a binding is a relation between names and their values. When a schema expression is used as a predicate, it must be used in a context where those names have been declared (if not, the schema predicate is ill-typed). Then the schema as predicate is *true* precisely when the values of those names in the current context satisfy the schema property, that is, if the current values of those names form one of the bindings in the set.

Syntax

```
Predicate ::= Expression
```

Examples

1. $Pythag == [\ l, m, n : \mathbb{N} \mid l * l = m * m + n * n\ ]$
   $S == [\ m, n : \mathbb{N} \mid P \land (\ \exists\, l : \mathbb{N} \bullet Pythag\ )\ ]$
   S denotes those bindings of m and n that satisfy all of:
   (a) the declaration constraint, of being natural numbers
   (b) the constraint P
   (c) the constraint that $m * m + n * n$ be the square of some natural number

Laws

Law 6.26 A schema as predicate is *true* precisely when the relevant current values from the local environment form one of the bindings.

$$[\dagger S] \vdash S \Leftrightarrow \theta S \in S$$

•

Relation predicate

Syntax

```
Predicate ::= Relation


Relation ::= Expression ∈ Expression
           | Expression = Expression
           | TemplateRelation
```

Definitions

Equality can be defined in terms of set membership. Two values are equal precisely when one is in the singleton set containing the other

$$[X]\ x, y : X \vdash x = y \Leftrightarrow x \in \{y\}$$

The meaning of an instantiated template relation can be defined in terms of set membership. The instantiated template is equivalent to the tuple of instantiating expressions being a member of the template relation.

$$[X]\ x : X;\ y : Y;\ \ldots;\ z : Z \vdash \\ T_1\ x\ T_2\ y \ldots z\ T_n \Leftrightarrow (x, y, \ldots, z) \in (T_1\ \_\ T_2\ \_ \ldots \_\ T_n)$$

Examples

1. $a \in \{a, b\}$
2. $a = b$
3. $1 < 2$
4. $(1, 2) \in (\_ < \_)$
5. $\neg$ finite prime
6. $\neg$ prime $\in$ (finite $\_$)
7. disjoint$\langle$prime, composite$\rangle$
8. $\langle$prime, composite$\rangle \in$ (disjoint $\_$)
9. $\langle\{0, 1\}$, prime, composite$\rangle$ partition $\mathbb{N}$
10. $(\langle\{0, 1\}$, prime, composite$\rangle, \mathbb{N} \in (_$ partition $\_)$

Laws

Law 6.27 Two sets are equal precisely when they have the same members

$$[X]\ a, b : \mathbb{P}\, X \vdash a = b \Leftrightarrow (\ \forall\, x : X \bullet x \in a \Leftrightarrow x \in b\ )$$

•

Schema Text

Syntax

```
SchemaText ::= [ DeclPart ] [ | Predicate ]
```

A *SchemaText* comprises an optional *DeclPart* and an optional *Predicate*.

The *DeclPart* declares variables and their types. If it is omitted (*ZRM*'s *SchemaText* does not allow it to be omitted), no variables are declared, and we get the set containing the empty binding, $\{(\!|\ \ |\!)\}$, if the *Predicate* is true, or the empty set if the *Predicate* is *false*.

The *Predicate* constrains the declared variables' values. If it is omitted, it is equivalent to *true*.

```
DeclPart ::= Declaration { ; Declaration }
```

The *DeclPart* declares variables and their types, in a semicolon separated list of *Declaration*s.

```
Declaration ::= DeclName { , DeclName } : Expression
              | EqualsDeclaration
              | Expression
```

Each *Declaration* is either a colon declaration, or an equals declaration, or an *Expression*. (*ZRM* restricts such an expression to be a schema reference; with *Standard* Z it can be *any* expression whose value is a set of bindings.)

Characteristic tuple

The characteristic tuple is the default value of an optional expression when omitted.

$$[\dagger S]$$
$$\chi : \mathbb{P}(\mathbb{P}\, S \times \theta S)$$

We provide a loose definition of this meta-operator, for the characteristic tuple of a schema, to allow the type-correct statement of some laws. (For a full definition, see the ISO Z Standard [ISO-Z 2002, §9.2].)

•

Chapter 7

Expressions

Expressions have types and values.

```
Expression ::= SchemaExpression
             | LambdaExpression
             | MuExpression
             | LetExpression
             | ConditionalExpression
             | PowerSet
             | CartesianProduct
             | TupleExpression
             | TupleComponentSelection
             | Application
             | Reference
             | SetExtension
             | SetComprehension
             | ( Expression )
```

SchemaExpressions are covered in the next chapter. PowerSet and CartesianProduct expressions are covered in §9.

Lambda expression

Intent

A lambda expression denotes a function (§21). The source elements are formed from the characteristic tuple of those values satisfying the schema text declaration. Each target element is given by the value of the expression in the context of the value of the corresponding source element.

Syntax

LambdaExpression ::= λ SchemaText $\bullet$ Expression

Examples

1. $succ = (\ \lambda x : \mathbb{N} \bullet x + 1\ ) = \{\ x : \mathbb{N} \bullet x \mapsto x + 1\ \}$
2. $swap = (\ \lambda x : X;\ y : Y \bullet y \mapsto x\ ) = \{\ x : X;\ y : Y \bullet (x, y) \mapsto (y, x)\ \}$
3. $S == [\ x : X;\ y : Y\ ];\ swap2 = (\ \lambda S \bullet y \mapsto x\ ) = \{\ S \bullet \theta S \mapsto (y, x)\ \}$

Laws

Law 7.1 A lambda function can be written as a set comprehension. (One reason for using a lambda expression is that it is guaranteed to be a *function*, not just a relation.)

$$[\dagger S]\ \vdash (\ \lambda S \bullet \mathcal{E}\ ) = \{\ S \bullet \chi\ S \mapsto \mathcal{E}\ \}$$

Law 7.2 The domain of a lambda function is the set of all values satisfying the schema text. (One reason for using a lambda function is that its domain is immediate.)

$$[\dagger S]\ \vdash \mathrm{dom}(\ \lambda S \bullet \mathcal{E}\ ) = \{\ S \bullet \chi\ S\ \} = S$$

$\bullet$

Mu expression

Intent

A mu expression denotes a value, or element. The element is given by the value of the expression in the context of the schema text, which itself must be unique (a singleton set of bindings, hence $\exists_1 S \bullet true$).

Syntax

```
MuExpression ::= μ SchemaText • Expression
               |  ( μ SchemaText )
```

Laws

Law 7.3 When the constructing expression is empty, the characteristic tuple is assumed.

$$[\dagger S] \vdash (\ \mu\, S\ ) = \mu\, S \bullet \chi\ S$$

•

Let expression

Intent

Introduce local declarations in an expression.

Syntax

```
LetExpression ::=
    let EqualsDeclaration { ; EqualsDeclaration } • Expression
```

Definition

The let expression is equivalent to a μ expression:

$$\begin{aligned} & \text{let}\, x_1 == e_1;\ \ldots;\ x_n == e_n \bullet E \\ \equiv\ & \mu\, x_1 == e_1;\ \ldots;\ x_n == e_n \bullet E \\ \equiv\ & \mu\, x_1 : \{e_1\};\ \ldots;\ x_n : \{e_n\} \bullet E \end{aligned}$$

•

Conditional expression

Intent

The value of the conditional expression is the value of the first sub-expression when the predicate is true, and it is the value of the second sub-expression when the predicate is false.

Syntax

```
ConditionalExpression ::=
    if Predicate then Expression else Expression
```

Laws

Law 7.4 A conditional expression can be written as an equivalent mu-expression.

$$[X \dagger P]\ a, b : \mathbb{P}\, X \vdash$$
$$(\ \text{if } P \text{ then } a \text{ else } b\ ) = (\ \mu\, x : \{a, b\} \mid P \land x = a \lor \neg\, P \land x = b\ )$$

Cartesian tuple expression

Intent

Construct a tuple. (This is isomorphic to the Schema binding extension, §8.)

Syntax

```
TupleExpression ::= ( Expression , Expression  { , Expression } )
```

Tuple component selection

Intent

Select the value of one component of a tuple. (Isomorphic to schema binding selection, §8.)

Syntax

```
TupleComponentSelection ::= Expression . NUMBER
```

Motivation

Consider the Cartesian product $a == X \times Y \times Z$. This denotes a set of *tuples*, one of which might be $t = (x_0, y_0, z_0)$.

With *ZRM* the direct way to select the y component of some tuple t is to write $(\ \lambda\, x : X;\ y : Y;\ z : Z \bullet y\ )t = y_0$, or to define this λ-expression as a named function and then use it.

This kind of selection is done quite often, so *Standard* Z has a new notation, allowing us to express this example as $t.2$. In general, if e denotes an expression whose type is that of a tuple, we can write $e.n$ to select the nth component of that tuple, where n is an unsigned base ten number literal whose value is between 1 and the number of components of the tuple. This new facility is just an abbreviation for the effective application of a λ-expression as given in the example above, which is why n must be a number literal, rather than any more general expression.

This dot notation does not allow us to *apply* component extraction. So we still need to define functions like *first* and *second* to do this. For example, if s is a sequence of pairs, $s : \mathrm{seq}(X \times Y)$, then $s \fatsemi \mathit{first}$ is the corresponding sequence of the first elements of those pairs, $\mathrm{seq}\, X$. We cannot write something like $s \fatsemi (\_.1)$ for this.

•

Function and generic application expression

Intent

Apply a function to its argument expressions, or instantiate a generic.

Syntax

```
Application
    ::= Expression Expression
     |  [ Expression ] NAME [ { Expression NAME } ] [ Expression ]
```

Examples

1. plain function application: $f\ x$
2. template function application: $x + y$
3. template generic application: $x \leftrightarrow y$

Reference expression

Intent

Refer to a variable name (possibly with any generic parameters explicitly instantiated), or a number literal.

Syntax

```
Reference ::= RefName
            |  RefName [ Expression { , Expression } ]
            |  NUMBER
```

Set extension expression

Intent

Write a set as an explicit list of its elements. (Also known as 'set display'.)

Syntax

```
SetExtension ::= { Expression { , Expression } }
```

Laws

Law 7.5 A set extension has an equivalent set comprehension.

$$[X]\ x_1, \ldots, x_n : X \vdash \{x_1, \ldots, x_n\} = \{\ x : X \mid \mathit{false} \lor x = x_1 \lor \ldots \lor x = x_n\ \}$$

•

Set comprehension expression

Intent

Define a set in terms of the properties of its elements.

Syntax

```
SetComprehension ::= { SchemaText [ • Expression ] }
```

Expanding the *SchemaText* into its two parts, we see that a set comprehension has three main components:

```
{ [ DeclPart ] [ | Predicate ] [ • Expression ] }
```

The *SchemaText* declares variables and their types (in its *DeclPart*) and constrains their values (in its *Predicate* part). The *Expression* is the constructing term, that constructs the elements that form the set from those variables' values.

All the parts are optional (except for two special cases noted at the end):

- If the *DeclPart* is empty, no new names are declared, so any names used in the other parts come from the surrounding environment.
- If the *Predicate* is empty, it is equivalent to *true*.
- If the *Expression* is empty, it is equivalent to the *characteristic tuple* of the *DeclPart*.
- If the *DeclPart* is the only part present, it is not permitted to be a single *Expression*, because {*Expression*} is viewed as a singleton set extension. To get a set comprehension of a single declaration expression, write { *Expression* | *true* }. (*ZRM* treats { *S* } as a set comprehension, and requires {(*S*)} for a set extension, see §2.4.1.)
- If all the optional parts are missing, all that remains is { }, the empty set. This is treated as a set extension.

Examples

- $\{\ x : \mathbb{Z} \mid x > 0 \bullet 2 * x\ \} = \{2, 4, 6, 8, \ldots\}$
- $\{\ x : \mathbb{Z} \bullet 2 * x\ \} = \{\ldots, -4, -2, 0, 2, 4, 6, \ldots\}$
- $\{\ x : \mathbb{Z} \mid x > 0\ \} = \{1, 2, 3, 4, 5, \ldots\}$
- $\{\ [\ x : \mathbb{Z}\ ] \mid x > 0 \bullet 2 * x\ \} = \{2, 4, 6, 8, \ldots\}$
- $\{\ [\ x : \mathbb{Z}\ ] \mid x > 0\ \} = \{(\!|\ x == 1\ |\!), (\!|\ x == 2\ |\!), (\!|\ x == 3\ |\!), \ldots\}$

Laws

Law 7.6 If the *DeclPart* is the only part present, the comprehension is equivalent to the declared set, or Cartesian products of the declared sets. If the *Predicate* is missing, it is equivalent to *true*. If the *Expression* is missing, it is equivalent to the *characteristic tuple* of the *Declaration*.

$$[X] \vdash \{\ x : X\ \} = X$$
$$[X, Y, Z] \vdash \{\ x : X;\ y : Y;\ z : Z\ \} = X \times Y \times Z$$
$$[\dagger S] \vdash \{\ S \bullet \mathcal{E}\ \} = \{\ S \mid \mathit{true} \bullet \mathcal{E}\ \}$$
$$[\dagger S, P] \vdash \{\ S \mid P\ \} = \{\ S \mid P \bullet \chi\ S\ \}$$

Law 7.7 x is a member of a set precisely when it is equal to one of the elements in the set.

$$[X \dagger S]\ x : X \vdash x \in \{\ S \bullet y\ \} \Leftrightarrow (\ \exists S \bullet x = y\ ) \qquad [x \text{ not declared in } S]$$

Law 7.8 Items in the declaration not needed to form the constructing term can be moved into the predicate and existentially quantified over.

$$[\dagger S, T, P] \vdash \{\ S;\ T \mid P \bullet \mathcal{E}\ \} = \{\ S \mid (\ \exists\, T \bullet P\ ) \bullet \mathcal{E}\ \} \quad [T \text{ not referenced in } \mathcal{E}]$$

———————————————•

Chapter 8

Schema Expressions

```
SchemaExpression ::= SchemaQuantification
                   | SchemaPropositional
                   | SchemaCombination
                   | SchemaRestriction
                   | SchemaRenamingExpression
                   | SchemaConstruction
                   | BindingConstruction
                   | BindingExtension
                   | BindingSelection
```

The value of a schema expression is the set of all bindings that obey its property.

In *ZRM*, schemas can be used as predicates and as expressions, but only in the form of schema *references*. As mentioned earlier, this restriction is particularly onerous in proof, because a schema reference cannot be expanded to its definition whilst remaining syntactically correct.

The restriction can also clutter specifications with intermediate names, introduced only to provide a name for a schema reference. For example, if there are two schemas S and T, and their composition is to be used to constrain the predicate part of a third schema, R, then an auxiliary schema needs to be introduced:

$$ScompT == S \fatsemi T$$
$$R == [\ \ldots \mid \mathcal{P};\ ScompT\ ]$$

With *Standard* Z, the syntax is more liberal, and so schema expressions can be used throughout. The above example can be written more simply as

$$R == [\ \ldots \mid \mathcal{P};\ S \fatsemi T\ ]$$

For example, consider a schema used to define an operation where most of the state stays unchanged. Suppose we have a schema with many state components:

$$S == [\ x_1, \ldots, x_n : X \mid \ldots\ ]$$

and we want to define an operation where only component x_i changes, in the way defined by some predicate $\mathcal{P}$, and all the others remain the same ($x'_j = x_j,\ j \neq i$). In *Standard* Z, we could write

$$Op_i == [\ \Delta S \mid \mathcal{P} \land \Xi S \setminus (x_i, x'_i)\ ]$$

Standard Z still does not provide for user-definable schema operators. Also, the use of Δ and Ξ are still *naming conventions*, not schema operators; we cannot write $\Xi(S \setminus (x_i))$.

What is permitted *Standard* Z by using schema expressions can also be said in *ZRM* by some other means, such as those illustrated above. The gain is in brevity and directness. Many of the ideas are explained in [Valentine 1995].

Schema quantification

Intent

Restrict the signature and bindings of a schema.

Syntax

```
SchemaQuantification ::= ∀ SchemaText • Expression
                       |  ∃ SchemaText • Expression
                       |  ∃1 SchemaText • Expression
```

Definition

The type of all the quantifications is the schema type of the *Expression* with all the names of the *SchemaText* hidden. The types of shared names must be compatible. *Standard* Z allows there to be names in the *SchemaText* not present in the *Expression*. (*ZRM* requires all names in the *SchemaText* to be present in the *Expression*.)

existential schema quantification:

The meaning of $\exists\, S \bullet T$ is defined as follows. Let b_s be a binding of S hiding any names that are not in T, with values given in the context of the surrounding specification.

Then the existential quantification expression is the set of bindings b of the correct type (as defined earlier), that can be extended with some binding b_s to form a binding in T:

$$\{\ b : \operatorname{dom} S \lhd T \mid \exists\, b_s : \operatorname{dom} T \lhd S \bullet b_s \cup b \in T\ \}$$

This definition cannot be expressed in Z itself. We illustrate it in a Z-like notation, as if we were using a model in which bindings were mappings from names to values. So we indicate the sets of schema names as $\operatorname{dom} S$ and $\operatorname{dom} T$, and use domain restriction to hide certain names. We use $b_1 \cup b_2$ to indicate the binding formed from combining the bindings b_1 and b_2; recall from the type condition that in this case b_s and b have no names in common.

unique existential schema quantification:

Using the same notation as above, the unique existential quantification expression is the set of bindings b of the correct type, that can be extended with precisely one of the bindings b_s to form a binding in T:

$$\{\ b : \operatorname{dom} S \lhd T \mid \exists_1\, b_s : \operatorname{dom} T \lhd S \bullet b_s \cup b \in T\ \}$$

universal schema quantification:

Using the same notation as above, the universal quantification expression is the set of bindings b of the correct type, that can be extended with every binding b_s to form a binding in T:

$$\{\ b : \operatorname{dom} S \lhd T \mid \forall\, b_s : \operatorname{dom} T \lhd S \bullet b_s \cup b \in T\ \}$$

Examples

1. existential schema quantification is used to hide names, whilst respecting the constraint.
2. Promotion pattern: $\mathit{GlobalOp} == \exists\, \Delta \mathit{Local} \bullet \Phi \mathit{Update} \land \mathit{LocalOp}$
3. Schema hiding: $S \setminus (x_1, \ldots, x_n) = \exists [\ x_1 : X_1;\ \ldots;\ x_n : X_n\ ] \bullet S$

Laws

Law 8.1 The quantified schema expressions can be converted to quantified predicates (where σ is the signature operator, §12)

$$[\dagger S, T] \vdash \exists S \bullet T = [\, \sigma T \setminus \sigma S \mid \exists S \bullet T \,]$$
$$[\dagger S, T] \vdash \exists_1 S \bullet T = [\, \sigma T \setminus \sigma S \mid \exists_1 S \bullet T \,]$$
$$[\dagger S, T] \vdash \forall S \bullet T = [\, \sigma T \setminus \sigma S \mid \forall S \bullet T \,]$$

Law 8.2 de Morgan's laws for schema quantifiers: negation pseudo-distributes through universal and existential schema quantification. (This law can be used to define existential schema quantification in terms of universal schema quantification, or vice versa.)

$$[\dagger S, T] \vdash \neg (\ \forall S \bullet T\ ) \Leftrightarrow (\ \exists S \bullet \neg\ T\ )$$
$$[\dagger S, T] \vdash \neg (\ \exists S \bullet T\ ) \Leftrightarrow (\ \forall S \bullet \neg\ T\ )$$

Law 8.3 Unique existential schema quantification can be written using existential and universal quantification. (Here it is assumed that S and T contain no dashed components; if they do, some other decoration should be used.)

$$[\dagger S, T] \vdash \exists_1 S \bullet T = (\ \exists S \bullet T\ ) \wedge (\ \forall S\,' \mid T\,' \bullet \theta S = \theta S\,'\ )$$

•

Schema propositional

Intent

Build schemas from component parts.

Syntax

```
SchemaPropositional ::= Expression ⇔ Expression
                      | Expression ⇒ Expression
                      | Expression ∨ Expression
                      | Expression ∧ Expression
                      | ¬ Expression
```

Definition

schema conjunction:

The type of the schema conjunction is the union of the types of the two expressions. The types of shared names must be compatible.

The meaning of $S \wedge T$ is defined as follows. Let b_s be a binding of S, and let b_t be a binding of T, with values given in the context of the surrounding specification. Then the schema conjunction expression is the set of bindings formed from b_s and b_t that are type correct:

$$\{\ b : \sigma_0(S;\ T) \mid \exists\, b_s : S;\ b_t : T \bullet b_s \cup b_t = b\ \}$$

This definition cannot be expressed in Z itself. We illustrate it in a Z-like notation, as if we were using a model in which bindings were mappings from names to values. The operator σ_0 is the analogue in this notation of the Z signature operator σ (§12): here σ_0 produces the set of all possible bindings with names drawn from S and from T. We use $b_1 \cup b_2$ to indicate the binding formed from combining the bindings b_1 and b_2; recall from the type condition that in this case any names b_s and b_t have in common are type compatible. Unions of bindings with different values for a given name are excluded, because they are not in b.

schema negation:

The type of the schema negation is the same type as the expression.

The meaning of $\neg\ S$ is defined as follows. Let b be all the bindings of S, with values given in the context of the surrounding specification. Then the schema negation expression is the set of bindings of the same type as b, but excluding all those in S:

$$[\dagger S] \vdash \{\ b : \sigma\ S \mid b \notin S\ \}$$

schema disjunction:

Schema disjunction can be defined in terms of conjunction and negation:

$$[\dagger S, T]\ \vdash S \vee T = \neg\ (\neg\ S \wedge \neg\ T)$$

schema implication:

Schema implication cab be defined in terms of disjunction and negation:

$$[\dagger S, T]\ \vdash S \Rightarrow T = \neg\ S \vee T$$

schema equivalence:

Schema equivalence is defined in terms of conjunction and implication:

$$[\dagger S, T] \vdash S \Leftrightarrow T = (S \Rightarrow T) \land (T \Rightarrow S)$$

Laws

Law 8.4 The schema negation can be converted to a predicate negation within a schema. Note this is not simply negation of the predicate part of S: any implicit predicates in the declaration part are also negated. This often results in a negated schema (and consequently disjunctions, implications and equivalences) having more bindings than 'expected'.

$$[\dagger S] \vdash \neg\, S = [\, \sigma S \mid \neg\, S \,]$$

Schema-as-Predicate ambiguity

If we have two schemas, S and T, and see a predicate such as $S \land T$, how should we interpret it? Is it the schema conjunction of S and T, used as a predicate: '$\mathsf{pred}(S \bigwedge T)$'? Or is it the logical conjunction of the predicates S and T: '$\mathsf{pred}\ S \land \mathsf{pred}\ T$'?

In practice, it does not matter which interpretation is used. The *Standard* Z scope rules and semantics have been carefully chosen so that, for all logical operators and their corresponding schema counterparts, the meanings are the same.

---•

Schema combination

Intent

Combine two operation schemas.

Syntax

```
SchemaCombination ::= Expression ⨟ Expression
                    |  Expression ≫ Expression
```

Definition

schema composition:

Schema composition $S \fatsemi T$ is designed for use with operation schemas, where dashed components of S (after states) are matched with undecorated components of T (before states). The types of corresponding matched components must be the same. The result is the conjunction of the schemas with the matched components identified and hidden. Any unmatched components of the same name in S and T must be of the same type and are merged. If the matched components are $x, \ldots, z$, then

$$[\dagger S, T] \vdash S \fatsemi T = (S[x_0/x', \ldots, z_0/z'];\ T[x_0/x, \ldots, z_0/z]) \setminus (x_0, \ldots, z_0)$$

schema piping:

Schema piping $S \gg T$ is designed for use with operation schemas, where components of S decorated with ! (outputs) are matched with components of T decorated with ? (inputs). The types of corresponding matched components must be the same. The result is the conjunction of the schemas with the matched components identified and hidden. Any unmatched components of the same name in S and T must be of the same type and are merged. If the matched components are $x, \ldots, z$, then

$$[\dagger S, T] \vdash S \gg T = (S[x_0/x!, \ldots, z_0/z!];\ T[x_0/x?, \ldots, z_0/z?]) \setminus (x_0, \ldots, z_0)$$

•

Schema restriction

Intent

Restrict the signature of a schema.

Syntax

```
SchemaRestriction ::= Expression \ ( DeclName { , DeclName } )
                    | Expression ⩤ Expression
                    | pre Expression
```

Definition

schema hiding:

$S \setminus (x_1, \ldots, x_n)$ has a signature equal to that of S minus the hidden names (the hidden names must be in the signature of S). If X_i is the type of x_i in the signature of S, then

$$[\dagger S] \vdash S \setminus (x_1, \ldots, x_n) = \exists [\ x_1 : X_1;\ \ldots;\ x_n : X_n\ ] \bullet S$$

schema projection:

$$[\dagger S, T] \vdash S \upharpoonright T = \{\ S;\ T \bullet \theta T\ \}$$

Projection restricts the schema $S \land T$ to the signature of T. It gives the set of bindings that conform to the signature of T, and whose values obey the constraints of both S and T.

schema precondition: $\operatorname{pre} S$ has the components of S that are decorated with a $'$ (after state) or ! (outputs) hidden. If A is the schema that is made from the signature of these components of S, then

$$[\dagger S] \vdash \operatorname{pre} S = \exists A \bullet S$$

Examples

1. $\{\langle\!|\ x == 1;\ y == 2\ |\!\rangle, \langle\!|\ x == 2;\ y == 2\ |\!\rangle\} \setminus (x) = \{\langle\!|\ y == 2\ |\!\rangle\}$
2. $\operatorname{pre}[\ \Delta S;\ x? : X;\ y! : Y\ ] = \exists S';\ y! : Y \bullet S$

•

Schema renaming expressions

Intent

Rename certain components in a schema.

Syntax

```
SchemaRenamingExpression
    ::= Expression [ DeclName / DeclName { , DeclName / DeclName } ]
     |  Expression STROKE
```

Definition

schema renaming:

$S[x_{new}/x_{old}, \ldots, z_{new}/z_{old}]$ must have no duplicate *old* names, but may have duplicate *new* names. The old names are replaced by the new names; any declarations that are merged by repeated new names must have the same type.

schema decoration:

The schema expression has every name renamed by that name with a decoration stroke added.

$$S\,^{\heartsuit} = S[x^{\heartsuit}/x, \ldots, z^{\heartsuit}/z]$$

Here $^{\heartsuit}$ is a stroke (a $'$, ?, !, or subscript digit).

Examples

1. $[\ x : X;\ y : Y \mid \mathcal{P}(x, y)\ ][z/x] = [\ z : X;\ y : Y \mid \mathcal{P}(z, y)\ ]$
2. $[\ x : X;\ y : Y \mid \mathcal{P}(x, y)\ ][z/x, w/y] = [\ z : X;\ w : Y \mid \mathcal{P}(z, w)\ ]$
3. $[\ x, y : X \mid \mathcal{P}(x, y)\ ][z/x, z/y] = [\ z : X \mid \mathcal{P}(z, z)\ ]$
4. $\{\langle\!|\ x == 1;\ y == 2\ |\!\rangle\}[z/x] = \{\langle\!|\ z == 1;\ y == 2\ |\!\rangle\}$
5. $[\ x : X;\ y : Y \mid \mathcal{P}(x, y)\ ]\,' = [\ x' : X;\ y' : Y \mid \mathcal{P}(x', y')\ ]$
6. $\{\langle\!|\ x == 1;\ y == 2\ |\!\rangle\}\,' = \{\langle\!|\ x' == 1;\ y' == 2\ |\!\rangle\}$

•

Schema construction

Intent

Construct a schema from variable declarations and predicates constraining their values.

Syntax

```
SchemaConstruction ::= [ SchemaText ]
```

The *SchemaText* is not permitted to be a single expression in this case.

Examples

1. $[\ x : \mathbb{N} \mid x < 2\ ] = \{ \langlebar\ x == 0\ \rangle, \langlebar\ x == 1\ \rangle \}$
2. $[\ x : X \mid P\ ] = \{\ x : X \mid P \bullet \langlebar\ x == x\ \rangle\ \}$
3. $[\ x, y : X \mid P\ ] = \{\ x, y : X \mid P \bullet \langlebar\ x == x, y == y\ \rangle\ \}$
4. The empty schema [ ] and its schema negation [\| *false*] act like boolean variables when used as predicates (see §12, Boolean expressions).
5. One use of a schema with an empty declaration part is to *name* a complicated predicate, which can then be used elsewhere.

Schema binding construction

Intent

Construct a binding from a schema and the values currently in scope. The schema expression provides the names; the environment provides the values of those names.

Syntax

```
BindingConstruction ::= θ Expression { STROKE }
```

Definition

The value of θE is the binding with names in the signature of the expression, and values equal to those names. Those values are taken from the environment, and must be type-compatible with the signature of the expression.

$$\theta E = \langle\!| \; x == x; \; \ldots; \; z == z \; |\!\rangle$$

The value of $\theta E\,'$ is the binding with names in the signature of the expression, and values equal to those of the decorated names.

$$\theta E\,' = \langle\!| \; x == x'; \; \ldots; \; z == z' \; |\!\rangle$$

Example

1. Promotion pattern: $\theta Local = global\ x?$
2. Ξ convention: $\Xi S == [\ \Delta S \mid \theta S = \theta S\,'\ ]$
3. The expression is a schema; the binding construction values need not obey the constraints of that schema, only its type:

$$S == [\ x : \mathbb{N}\ ]$$

$$\begin{array}{|l} x == -3; \; x' == -5 \\ \hline \theta S = \langle\!| \; x == -3 \; |\!\rangle \\ \theta S\,' = \langle\!| \; x == -5 \; |\!\rangle \end{array}$$

•

Schema binding extension

Intent

Write a single binding as an explicit binding of names to their values. (Isomorphic to Cartesian tuple expression, §7.)

Syntax

```
BindingExtension ::=
    ⦇ DeclName == Expression { ; DeclName == Expression } ⦈
```

Motivation

Consider the schema $S == [\ a : X \times Y;\ b : \mathbb{F}\, X \mid a.1 \in b\ ]$. This denotes a set of *bindings*, all those values that satisfy the schema constraints (both those constraints implicit in the declaration, and those explicit in the predicate part).

Assume that we want to define a particular element s of the set of all bindings given by S. With *ZRM* we would do this using a μ-expression:

$$\mu\, s : S \mid s.a = (x, y) \land s.b = \{x, z\}$$
$$\mu\, s : [\ S \mid a = (x, y) \land b = \{x, z\}\ ]$$
$$\mu\, s : [\ a : \{(x, y)\};\ b : \{\{x, z\}\}\ ]$$

With the *Standard* Z '==' notation, this last variant can be expressed more neatly as

$$\mu\, s : [\ a == (x, y), b == \{x, z\}\ ]$$

or, slightly more briefly, using the binding notation, as

$$⦇\ a == (x, y), b == \{x, z\}\ ⦈$$

In specifications, however, explicit bindings are seldom the best way to write such an expression. For example, $t \in S$, where t is a binding, can equally well be expressed as $\exists\, S \bullet T$, where T is the conjunct of the equalities corresponding to t, as in:

$$⦇\ a == (x, y), b == \{x, z\}\ ⦈ \in S$$
$$\exists\, S \bullet a = (x, y) \land b = \{x, z\}$$

The latter form has the advantage that any number of components may be constrained, in any way, whereas the former requires them each to be given an explicit value.

Explicit bindings *are* useful intermediate objects in formal proofs.

Example: leap years

The Gregorian calendar, introduced under the authority of Pope Gregory XIII, replaced the Julian calendar, introduced under the authority of Julius Caesar, which had used the simpler leap year algorithm of 'divisible by four'.

The new calendar was adopted at different times throughout the world; in those parts where it was adopted first, 4th October 1582 (Julian) was followed by 15th October 1582 (Gregorian). It was introduced in the British Empire in 1752 (the English monarch had not been on good terms with the Pope in 1582) by which time an adjustment of 11 days was needed because of counting 1700 as a leap year. This sparked riots from people who had to pay their next quarter's rents and taxes sooner: "give us back our 11 days". (Popular legend would have us believe that our forebears rioted because they naively thought that they had had 11 days of their lives stolen. However, they were not at all naive: they noticed that they had had some of their *money* stolen!)

Because of the gap in numbering when the change occurred, and because the change occurred at different times in different places (Greece, for example, adopted the new calendar in 1923), calculating dates in the past may require one to specify which calendar is being used. For example, in 1917 the 'October Revolution' in Russia, still using the Julian calendar, was perceived in western Europe as taking place in November.

[Reingold & Dershowitz 2001] is a good source of information about the intricacies of calendars.

According to the Gregorian calendar, a *leap year* is any year divisible by four, unless it is divisible by 100, unless it is also divisible by 400.

$$
\begin{array}{|l}
\hline
\mathit{LeapYear} \\
\hline
year : \mathbb{N}_1 \\
\hline
year \in (\{\ n : \mathbb{N} \bullet 4 * n\ \} \setminus \{\ n : \mathbb{N} \bullet n * 100\ \}) \cup \{\ n : \mathbb{N} \bullet n * 400\ \} \\
\hline
\end{array}
$$

We require *year* to be positive. There was no year zero: year numbers move inconveniently from 1 BC directly to AD 1 (the absence of a year zero explains why the new millennium started in AD 2001).

$$MONTH ::= jan \mid feb \mid mar \mid apr \mid may \mid jun \mid jly \mid aug \mid sep \mid oct \mid nov \mid dec$$

$$monthName == \langle jan, feb, mar, apr, may, jun, jly, aug, sep, oct, nov, dec \rangle$$

Date takes into account the variable number of days in a month, including leap years:

$$
\begin{array}{|l}
\hline
\textit{Date} \\
\textit{day} : 1 \mathinner{..} 31 \\
\textit{month} : 1 \mathinner{..} 12 \\
\textit{year} : \mathbb{N}_1 \\
\hline
\exists\, \textit{daysInMonth} == \langle 31, \text{if } \textit{LeapYear} \text{ then } 29 \text{ else } 28, \\
\qquad\qquad 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 \rangle \bullet \\
\quad \textit{day} \leq \textit{daysInMonth}\ \textit{month} \\
\hline
\end{array}
$$

Hence 1995 is not a leap year (not divisible by four), 1996 is leap year (divisible by four), 1900 is not a leap year (divisible by 100), and 2000 is a leap year (divisible by 400).

$$\vdash \langle\!|\ \textit{day} == 29, \textit{month} == 2, \textit{year} == 1995\ |\!\rangle \notin \textit{Date}$$
$$\vdash \langle\!|\ \textit{day} == 29, \textit{month} == 2, \textit{year} == 1996\ |\!\rangle \in \textit{Date}$$
$$\vdash \langle\!|\ \textit{day} == 29, \textit{month} == 2, \textit{year} == 1900\ |\!\rangle \notin \textit{Date}$$
$$\vdash \langle\!|\ \textit{day} == 29, \textit{month} == 2, \textit{year} == 2000\ |\!\rangle \in \textit{Date}$$

If we wished to put the *daysInMonth* component into the signature, maybe for use elsewhere, we would have to recast these examples using the existential form:

$$\vdash \exists\, \textit{Date} \bullet \textit{day} = 29 \land \textit{month} = 2 \land \textit{year} = 2000$$

•

Binding selection

Intent

Select the value of one component of a binding. (Isomorphic to cartesian tuple component selection, §7.)

Syntax

```
BindingSelection ::= Expression . RefName
```

Definition

In $S.n$, S must be a binding, and n must be one of its names. The value of the resulting expression is the value of that name in the binding.

Examples

1. $\langle\!\langle\ x == 2, y == 3\ \rangle\!\rangle.y = 3$
2. $\langle\!\langle\ day == 29, month == 2, year == 1995\ \rangle\!\rangle.year = 1995$
3. The expression does not have to be a binding extension (but it must be binding-valued):

$$S == [\ x, y : \mathbb{N}\ ]$$

$$\begin{array}{|l} x == 3;\ y' == 5 \\ \hline (\theta S).x = 3 \\ (\theta S\ ').y = 5 \end{array}$$

Chapter 9

Type constructors

There are four ways to introduce new types into a Z specification

- given set, including free type
- power set
- Cartesian product
- schema type

Given Set Paragraph

Intent

Introduce a new basic type about which nothing more is (yet) known. Parameterise the specification by this set.

Syntax

```
GivenSet ::= [ NAME { , NAME } ]
```

Examples

1. $[ID, CHAR, NAME]$

Free Type Paragraph

Intent

Introduce a new basic type along with some structure.

Syntax

```
FreeType ::= NAME ::= Branch { | Branch }
             { &
             NAME ::= Branch { | Branch } }

Branch ::= DeclName [ ⟪ Expression ⟫ ]
```

Examples

1. enumerated type:

   $$DAY ::= sun \mid mon \mid tues \mid wed \mid thurs \mid fri \mid sat$$

2. union type:

   $$CARD ::= pips\langle\langle 1 \mathinner{..} 10\rangle\rangle \mid jack \mid queen \mid king$$

3. simple recursive free type:

   $$TREE ::= leaf \mid node\langle\langle TREE \times \mathbb{N} \times TREE\rangle\rangle$$

4. mutually recursive free type:

   $$DECL ::= decl\langle\langle ID \times EXPR\rangle\rangle$$
   $$\&$$
   $$EXPR ::= letExpr\langle\langle \mathrm{seq}\, DECL \times EXPR\rangle\rangle \mid num\langle\langle \mathbb{N}\rangle\rangle$$

Laws

Law 9.1 The induction principle, for enumerated types.

The free type definition

$$N ::= a_1 \mid \ldots \mid a_m$$

is semantically equivalent to

$$[N]$$

$$\begin{array}{|l} a_1, \ldots, a_m : N \\ \hline \langle \{a_1\}, \ldots, \{a_m\} \rangle \text{ partition } N \end{array}$$

The a_i are a collection of *distinct* names of type N ('no confusion'), and they together comprise the *whole* of N ('no junk').

Example: The DAY example above is semantically equivalent to

$$[DAY]$$

$$\begin{array}{|l} sun, \ldots, sat : DAY \\ \hline \langle \{sun\}, \{mon\}, \{tues\}, \{wed\}, \{thurs\}, \{fri\}, \{sat\} \rangle \text{ partition } DAY \end{array}$$

Law 9.2 The induction principle, for union types.

The free type definition

$$N ::= a_1 \mid \ldots \mid a_m \mid b_1 \langle\langle E_1 \rangle\rangle \mid \ldots \mid b_n \langle\langle E_n \rangle\rangle$$

where there is no recursion (the E_i do not refer to N), is semantically equivalent to

$$[N]$$

$$\begin{array}{|l} a_1, \ldots, a_m : N \\ b_1 : E_1 \rightarrowtail N ; \ \ldots ; \ b_n : E_n \rightarrowtail N \\ \hline \langle \{a_1\}, \ldots, \{a_m\}, \text{ran } b_1, \ldots, \text{ran } b_n \rangle \text{ partition } N \end{array}$$

The a_i are a collection of *distinct* names of type N. The b_j are a collection of injections from expressions to *distinct* elements of N. Together these elements of N comprise the *whole* of N.

Example: The *CARD* example above is semantically equivalent to:

$$[CARD]$$

$$\begin{array}{|l} jack, queen, king : CARD \\ pips : 1 \mathinner{..} 10 \rightarrowtail CARD \\ \hline \langle \{jack\}, \{queen\}, \{king\}, \operatorname{ran} pips \rangle \operatorname{partition} CARD \end{array}$$

Law 9.3 The induction principle, for simple recursive free types.

The free type definition

$$N ::= a_1 \mid \ldots \mid a_m \mid b_1 \langle\!\langle E_1 \rangle\!\rangle \mid \ldots \mid b_n \langle\!\langle E_n \rangle\!\rangle$$

is semantically equivalent to

$$[N]$$

$$\begin{array}{|l} a_1, \ldots, a_m : N \\ b_1 : E_1 \rightarrowtail N ; \ \ldots ; \ b_n : E_n \rightarrowtail N \\ \hline \operatorname{disjoint} \langle \{a_1\}, \ldots, \{a_m\}, \operatorname{ran} b_1, \ldots, \operatorname{ran} b_n \rangle \\ \forall\, w : \mathbb{P}\, N \mid \\ \qquad \{a_1, \ldots, a_m\} \\ \qquad \cup\, b_1(\!|\, \operatorname{let} N == w \bullet E_1 \,|\!) \\ \qquad \cup \ldots \\ \qquad \cup\, b_n(\!|\, \operatorname{let} N == w \bullet E_n \,|\!) \\ \quad \subseteq w \\ \bullet\ w = N \end{array}$$

The a_i are a collection of *distinct* names of type N. The b_j are a collection of injections from expressions to *distinct* elements of N. Together these elements of N comprise the *whole* of N.

Example: The *TREE* example above is semantically equivalent to (after some simplification):

$$[TREE]$$

$$\begin{array}{|l} leaf : TREE \\ node : TREE \times \mathbb{N} \times TREE \rightarrowtail TREE \\ \hline \operatorname{disjoint} \langle \{leaf\}, \operatorname{ran} node \rangle \\ \forall\, w : \mathbb{P}\, TREE \mid \\ \qquad \{leaf\} \cup node(\!|\, w \times \mathbb{N} \times w \,|\!) \subseteq w \\ \bullet\ w = TREE \end{array}$$

Law 9.4 The full induction principle, for mutually recursive free types.

The free type definition

$$
\begin{array}{l}
N_1 ::= a_{11} \mid \ldots \mid a_{1m_1} \mid b_{11} \langle\langle E_{11} \rangle\rangle \mid \ldots \mid b_{1n_1} \langle\langle E_{1n_1} \rangle\rangle \\
\& \ldots \& \\
N_r ::= a_{r1} \mid \ldots \mid a_{rm_r} \mid b_{r1} \langle\langle E_{r1} \rangle\rangle \mid \ldots \mid b_{rn_r} \langle\langle E_{rn_r} \rangle\rangle
\end{array}
$$

is semantically equivalent to

$$[N_1, \ldots, N_r]$$

$$
\begin{array}{|l}
a_{11}, \ldots, a_{1m_1} : N_1 \\
\ldots \\
a_{r1}, \ldots, a_{rm_r} : N_r \\
b_{11} : E_{11} \rightarrowtail N_1;\ \ldots;\ b_{1n_1} : E_{1n_1} \rightarrowtail N_1 \\
\ldots \\
b_{r1} : E_{r1} \rightarrowtail N_r;\ \ldots;\ b_{rn_r} : E_{rn_r} \rightarrowtail N_r \\
\hline
\mathrm{disjoint} \langle \{a_{11}\}, \ldots, \{a_{1m_1}\}, \mathrm{ran}\ b_{11}, \ldots, \mathrm{ran}\ b_{1n_1} \rangle \\
\ldots \\
\mathrm{disjoint} \langle \{a_{r1}\}, \ldots, \{a_{rm_r}\}, \mathrm{ran}\ b_{r1}, \ldots, \mathrm{ran}\ b_{rn_r} \rangle \\
\forall\, w_1 : \mathbb{P}\, N_1;\ \ldots;\ w_r : \mathbb{P}\, N_r \mid \\
\qquad \{a_{11}, \ldots, a_{1m_1}\} \\
\qquad \cup\, b_{11}(\!|\ \mathrm{let}\ N_1 == w_1;\ \ldots;\ \mathrm{let}\ N_r == w_r \bullet E_{11}\ |\!) \\
\qquad \cup \ldots \\
\qquad \cup\, b_{1n_1}(\!|\ \mathrm{let}\ N_1 == w_1;\ \ldots;\ \mathrm{let}\ N_r == w_r \bullet E_{1n_1}\ |\!) \\
\quad \subseteq w_1 \\
\quad \land \ldots \land \\
\qquad \{a_{r1}, \ldots, a_{rm_r}\} \\
\qquad \cup\, b_{r1}(\!|\ \mathrm{let}\ N_1 == w_1;\ \ldots;\ \mathrm{let}\ N_r == w_r \bullet E_{11}\ |\!) \\
\qquad \cup \ldots \\
\qquad \cup\, b_{rn_r}(\!|\ \mathrm{let}\ N_1 == w_1;\ \ldots;\ \mathrm{let}\ N_r == w_r \bullet E_{rn_r}\ |\!) \\
\quad \subseteq w_r \\
\bullet\ w_1 = N_1 \land \ldots \land w_r = N_r
\end{array}
$$

Example: The *DECL*, *EXPR* example above is semantically equivalent to (after some simplification):

$$[DECL, EXPR]$$

$$
\begin{array}{l}
decl : ID \times EXPR \rightarrowtail DECL \\
letExpr : \operatorname{seq} DECL \times EXPR \rightarrowtail EXPR;\ num : \mathbb{N} \rightarrowtail EXPR \\
\hline
\operatorname{disjoint}\langle \operatorname{ran} letExpr, \operatorname{ran} num \rangle \\
\forall\, w_d : \mathbb{P}\, DECL;\ w_e : \mathbb{P}\, EXPR \mid \\
\qquad decl(\!|\ ID \times w_e\ |\!) \subseteq w_d \\
\qquad \land\ letExpr(\!|\ \operatorname{seq} w_d \times w_e\ |\!) \cup num(\!|\ \mathbb{N}\ |\!) \subseteq w_e \\
\qquad \bullet\ w_d = DECL \land w_e = EXPR
\end{array}
$$

Power set Expression

Intent

The value of a power set expression is the set of all subsets of its argument set.

Syntax

```
PowerSet ::= ℙ Expression
```

Examples

1. $[X] \vdash \mathbb{P}\, \varnothing[X] = \{\varnothing\}$
2. $\vdash \mathbb{P}\{1\} = \{\varnothing, \{1\}\}$
3. $\vdash \mathbb{P}\{1,2\} = \{\varnothing, \{1\}, \{2\}, \{1,2\}\}$
4. $\vdash \mathbb{P}\{1,2,3\} = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
5. If we want to state in a predicate that a is a set of elements drawn from X, we can say either $a \subseteq X$ or $a \in \mathbb{P}\, X$. If we want to *declare* that a is such a set, we have notation for the latter option only, $a : \mathbb{P}\, X$.
6. $\vdash even \in \mathbb{P}\, \mathbb{N}$
7. $\vdash \{even, \text{prime}\} \in \mathbb{P}\, \mathbb{P}\, \mathbb{N}$; $\vdash \{even, odd\} \in \mathbb{P}\, \mathbb{P}\, \mathbb{N}$
8. $\vdash \{\{even, odd\}, \{\text{prime}, \text{composite}\}\} \in \mathbb{P}\, \mathbb{P}\, \mathbb{P}\, \mathbb{N}$

Laws

Law 9.5 A power set is finite precisely when the set is finite.

$$[X] \ \vdash \text{finite}(\mathbb{P}\, X) \Leftrightarrow \text{finite}\, X$$

Law 9.6 The size of a power set of a finite set is 2 raised to the power of the size of the set.

$$[X] \ a : \mathbb{F}\, X \vdash \#(\mathbb{P}\, a) = 2 ** \#a$$

Law 9.7 The power set of a set a is the set containing all the subsets of a.

$$[X] \ a : \mathbb{P}\, X \vdash \mathbb{P}\, a = \{\ b : \mathbb{P}\, X \mid b \subseteq a\ \}$$

Hence the power set of any set, including the empty set, contains at least one element, the empty set: $\varnothing \in \mathbb{P}\, a$. The power set of any set also contains that set: $a \in \mathbb{P}\, a$.

Law 9.8 Any element of a subset is an element of the containing set: if a is a subset of b and x is in a, then x is in b.

$$[X] \ a, b : \mathbb{P}\, X;\ x : X \mid a \in \mathbb{P}\, b \land x \in a \vdash x \in b$$

Power set does not in general distribute through union.

For example

$$\begin{array}{l}[X] \ x, y : X \vdash \\ \quad \mathbb{P}\{x\} \cup \mathbb{P}\{y\} = \{\varnothing, \{x\}\} \cup \{\varnothing, \{y\}\} = \{\varnothing, \{x\}, \{y\}\} \\ \quad \land\ \mathbb{P}(\{x\} \cup \{y\}) = \mathbb{P}\{x, y\} = \{\varnothing, \{x\}, \{y\}, \{x, y\}\}\end{array}$$

Law 9.9 Power set distributes through union precisely when one of the arguments is the union of all the arguments.

$$[X] \ \alpha : \mathbb{P}\,\mathbb{P}\, X \vdash \bigcup\{\ a : \alpha \bullet \mathbb{P}\, a\ \} = \mathbb{P}(\bigcup \alpha) \Leftrightarrow \bigcup \alpha \in \alpha$$
$$[X] \ a, b : \mathbb{P}\, X \vdash \mathbb{P}\, a \cup \mathbb{P}\, b = \mathbb{P}(a \cup b) \Leftrightarrow a \cup b \in \{a, b\}$$

■ **Law 9.10** Power set distributes through intersection.

$$[X] \ \alpha : \mathbb{P}\,\mathbb{P}\, X \vdash \mathbb{P}(\bigcap \alpha) = \bigcap\{\ a : \alpha \bullet \mathbb{P}\, a\ \}$$
$$[X] \ a, b : \mathbb{P}\, X \vdash \mathbb{P}(a \cap b) = \mathbb{P}\, a \cap \mathbb{P}\, b$$

Law 9.11 Hence power set is subset order-preserving (§26). a is a subset of b precisely when a's power set is a subset of b's power set.

$$[X]\ a, b : \mathbb{P}\, X \vdash a \subseteq b \Leftrightarrow \mathbb{P}\, a \subseteq \mathbb{P}\, b$$

Law 9.12 Specialising the order-preserving law about upper bounds (law 26.36) to $f = \mathbb{P}\_$, gives

$$[X]\ \alpha : \mathbb{P}\,\mathbb{P}\, X \vdash \bigcup\{\ a : \alpha \bullet \mathbb{P}\, a\ \} \subseteq \mathbb{P}(\bigcup \alpha)$$
$$[X]\ a, b : \mathbb{P}\, X \vdash \mathbb{P}\, a \cup \mathbb{P}\, b \subseteq \mathbb{P}(a \cup b)$$

■ **Law 9.13** Power set never distributes through set difference or symmetric set difference.

$$[X]\ a, b : \mathbb{P}\, X \vdash \mathbb{P}\, a \setminus \mathbb{P}\, b \neq \mathbb{P}(a \setminus b)$$
$$[X]\ a, b : \mathbb{P}\, X \vdash \mathbb{P}\, a \ominus \mathbb{P}\, b \neq \mathbb{P}(a \ominus b)$$

———————————•

Cartesian product Expression

Intent

The value of a Cartesian product expression is the set of all tuples of its argument sets.

Syntax

```
CartesianProduct ::=
            Expression × Expression { × Expression }
```

The cross product expression must contain at least two sub-expressions.

The Cartesian product construction can be used with any number of components from two upwards. Each gives a distinct family of constructions, and is not the same as an iterated use of Cartesian pair. Thus the set of triples $a \times b \times c$, the set of pairs whose second component is a pair $a \times (b \times c)$, and the set of pairs whose first component is a pair $(a \times b) \times c$, are all distinct from each other, and are all of different types.

They are, however, isomorphic, in that it is possible to define a property-preserving bijection (§21) between the various types. For example, $(\ \lambda\, x : X;\ y : Y;\ z : Z \bullet ((x, y), z)\ )$ is a bijection, $X \times Y \times Z \rightarrowtail\!\!\!\!\!\twoheadrightarrow (X \times Y) \times Z$. Similar bijections can be defined for the other cases.

Examples

1. $x : X \vdash \{x\} \times \varnothing[Y] = \varnothing[X \times Y]$
2. $\{x\} \times \{y\} = \{(x, y)\}$
3. $\{x\} \times \{y\} \times \{z\} = \{(x, y, z)\}$
4. $\{x\} \times \{y, y'\} = \{(x, y), (x, y')\}$
5. $\{x\} \times \{y, y'\} \times \{z\} = \{(x, y, z), (x, y', z)\}$
6. $\{x, x'\} \times \{y, y'\} = \{(x, y), (x, y'), (x', y), (x', y')\}$
7. Cartesian pairs are used to model relations (§16) and functions (§21).

Laws about Cartesian pairs

Law 9.14 Laws about the second argument of a Cartesian pair can be derived as 'duals' of laws about the first argument, by using the fact that $X \times Y$ is isomorphic to $Y \times X$ under the bijection:

$$[X, Y] \ \vdash (\ \lambda\, x : X;\ y : Y \bullet (y, x)\ ) \in X \times Y \rightarrowtail\!\!\!\!\!\twoheadrightarrow Y \times X$$

Law 9.15 The Cartesian pair contains all tuples formed from elements of its argument sets.

$$[X, Y] \ \vdash X \times Y = \{\ x : X;\ y : Y\ \}$$

This equation could equally well be written as

$$X \times Y = \{\ x : X;\ y : Y \bullet (x, y)\ \}$$

The *characteristic tuple*, the term after the '•', is usually elided (§7).

Law 9.16 The Cartesian product of two sets is finite precisely when each of its components sets is finite or if either set is empty.

$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash \\ \quad \mathrm{finite}(a \times b) \Leftrightarrow \mathrm{finite}\, a \wedge \mathrm{finite}\, b \vee a = \varnothing \vee b = \varnothing$$

Law 9.17 The size of a Cartesian product with finite components is the product of the component sizes.

$$[X, Y]\ a : \mathbb{F}\, X;\ b : \mathbb{F}\, Y \vdash \#(a \times b) = \#a * \#b$$

Law 9.18 The Cartesian pair is empty precisely when at least one of the product sets is empty.

$$[X, Y]\ \vdash X \times Y = \varnothing \Leftrightarrow X = \varnothing \vee Y = \varnothing$$

Law 9.19 Cartesian pair distributes through set difference (on both arguments).

$$[X, Y]\ a, b : \mathbb{P}\, X \vdash (a \setminus b) \times Y = (a \times Y) \setminus (b \times Y)$$

Law 9.20 So law 13.45 gives us that Cartesian pair also distributes through set union, intersection, and symmetric set difference (on both arguments). Cartesian pair also distributes through generalised union and non-empty generalised intersection.

$$[X, Y]\ \alpha : \mathbb{P}_1\, \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash \bigcap \alpha\ \times b = \bigcap \{\ a : \alpha \bullet a \times b\ \}$$
$$[X, Y]\ a, a' : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash (a \cap a') \times b = (a \times b) \cap (a' \times b)$$

$$[X, Y]\ \alpha : \mathbb{P}\, \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash \bigcup \alpha\ \times b = \bigcup \{\ a : \alpha \bullet a \times b\ \}$$
$$[X, Y]\ a, a' : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash (a \cup a') \times b = (a \times b) \cup (a' \times b)$$

$$[X, Y]\ a, a' : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash (a \ominus a') \times b = (a \times b) \ominus (a' \times b)$$

The intersection distribution does not hold in general for empty α.

For example

$$[X, Y]\ b : \mathbb{P}\, Y \vdash \\ \quad \bigcap \varnothing[\mathbb{P}\, X]\ \times b = X \times b \\ \quad \wedge \bigcap \{\ a : \varnothing[\mathbb{P}\, X] \bullet a \times b\ \} = \bigcap \varnothing[\mathbb{P}(X \times Y)] = X \times Y$$

Law 9.21 The distribution properties give that Cartesian pair is subset-order preserving (section 26) on both arguments.

$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash a \times b \subseteq X \times Y$$

Law 9.22 Two pairs are equal precisely when their corresponding components are equal.

$$[X, Y]\ x, x' : X;\ y, y' : Y \vdash (x, y) = (x', y') \Leftrightarrow x = x' \wedge y = y'$$

Laws about Cartesian tuples

All the laws given above for Cartesian pairs generalise to Cartesian tuples of any number of components, because of the isomorphisms between $(X \times Y) \times Z$ and $X \times Y \times Z$, and so on. So, for example, law 9.18 generalises (informally) to

$$X \times \ldots \times Z = \varnothing \Leftrightarrow X = \varnothing \vee \ldots \vee Z = \varnothing$$

There is no way in Z of formally describing the concept 'all Cartesian products', no way of formalising the '. . . ' above. So Z has no way of making a formal statement of the complete family of laws. We state the laws for Cartesian pairs above; the generalisation to arbitrary tuples is clear from the relevant isomorphism.

---•

Schema type

Intent

Like Cartesian product, the schema type is also a product type, with any number of named components from zero upwards. Whereas the Cartesian product can be thought of as constructing a product by labelling the components with their position in the product, a schema can be thought of as constructing a product by labelling the components with names.

Syntax

Schema types are constructed in the various schema expressions defined in §8.

Laws

There is an isomorphism between schema types (with two or more components) and Cartesian products. For example, $(\lambda x : X;\ y : Y \bullet \langle\!| s_x == x, s_y == y |\!\rangle)$ is a bijection, $X \times Y \rightarrowtail\!\!\!\!\!\rightarrow [\ s_x : X;\ s_y : Y\ ]$. Hence laws about Cartesian products can also be applied to schemas. For example

Law 9.23 The schema analogue of law 9.18 is as follows. The schema type is empty precisely when at least one of the product sets is empty.

$$[X, Y] \vdash [\ s_x : X;\ s_y : Y\ ] = \varnothing \Leftrightarrow X = \varnothing \lor Y = \varnothing$$

There is no way in Z of formally describing the concept 'all schema types', and this inability is in some sense even worse than the inability to describe general Cartesian products. The example law above uses *specific* component names s_x and s_y, and formally there is no implication that the law applies if different component names are used. However, again we are saved by an isomorphism: $[\ s_x : X;\ s_y : Y\ ]$ is isomorphic to $[\ t_x : X;\ t_y : Y\ ]$.

So although there is no way of making a formal statement of laws for all schema types, the generalisation of the Cartesian product laws to arbitrary schemas is clear from the relevant isomorphism.

We give here some of the more important laws in explicit two-component schema form.

Law 9.24 The schema analogue of law 9.15 is as follows. The schema type contains all bindings formed from elements of its argument sets.

$$[X, Y] \vdash [\ s_x : X;\ s_y : Y\ ] = \{\ x : X;\ y : Y \bullet \langle\!| s_x == x, s_y == y |\!\rangle\ \}$$

Law 9.25 The schema analogue of law 9.16 is as follows. A schema is finite (is a finite set of bindings) when each of its component sets is finite or if any component set is empty.

$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash$$
$$\quad \mathrm{finite}[s_x : a;\ s_y : b] \Leftrightarrow \mathrm{finite}\, a \land \mathrm{finite}\, b \lor a = \varnothing \lor b = \varnothing$$

Law 9.26 The schema analogue of law 9.17 is as follows. The size of an unconstrained schema with finite components is the product of the component sizes.

$$[X, Y]\ a : \mathbb{F}\, X;\ b : \mathbb{F}\, Y \vdash \#[\ s_x : a;\ s_y : b\ ] = \#a * \#b$$

Law 9.27 The schema analogue of law 9.19 is as follows. Schema type distributes through set difference. (And so law 13.45 gives us that schema type also distributes through set union, intersection, and symmetric set difference.)

$$[X, Y]\ a, b : \mathbb{P}\, X \vdash [\ s_x : a \setminus b;\ y : Y\ ] = [\ s_x : a;\ y : Y\ ] \setminus [\ s_x : b;\ y : Y\ ]$$

———————————————•

Chapter 10

Lexis

10.1 Introduction

Standard Z provides a large amount of freedom in defining new names, including mixing letters and special symbols, and provision for subscripts and superscripts. We need some recognition that Z, when typeset, uses non-ascii characters such as Ξ, $\oplus$, $\rightarrowtail$, sub- and super-scripts, and schema boxes.

10.2 Z characters

Standard Z names are built up from Unicode characters [Unicode 1996], called `ZCHAR`s. There are four classes of `ZCHAR`s, based on Unicode properties. A basic set of characters is provided, but this set can be extended with any Unicode character as desired.

- `DIGIT`: `0 .. 9`; extensible with any Unicode character with the *Number* property
- `LETTER`: all upper and lower case Latin and Greek alphabetic characters, 'fat' letters $\mathbb{A}$, $\mathbb{Z}$, and $\mathbb{P}$; extensible with any Unicode character with the *Letter* property
- `SPECIAL`: all of `STROKECHAR`, `WORDGLUE`, brackets ( ) [ ] { }, schema box characters, newline, and space
- `SYMBOL`: mathematical symbols (all other characters, including any user-defined ones); extensible with any other Unicode character not already captured in another class

where the character subclasses are

- `STROKECHAR`: dash ′, question mark (or query) ?, and exclamation mark (or shriek, or pling) !
- `WORDGLUE`: underscore _, begin and end superscript markers ↘ and ↖, and begin and end subscript markers ↗ and ↙

These `ZCHAR`s are the basic characters used to define *Standard* Z words. The ISO Z Standard [ISO-Z 2002] defines certain ascii *markups*: mappings between strings of ascii characters and `ZCHAR`s. For example, the LaTeX markup defines "`\power`" → '$\mathbb{P}$'.

10.3 Z words

`ZCHAR`s are juxtaposed to form names, with certain restrictions. The aim here is to minimise the amount of white space that needs to be input. We want the three character string 'x', 'a', 'y' to be the single identifier 'xay', but the string 'x', '+', 'y' to be the expression '$x + y$'. This motivates the separation of characters into letters and symbols: if a letter occurs next to a symbol, they are assumed to be in different names. Letters and symbols can be used in the same name by separating them with a `WORDGLUE` character. The full syntax for a *Standard* Z word is:

```
AlphaNum  ::= LETTER | DIGIT
WordPart  ::= { AlphaNum } | { SYMBOL }
WordPart1 ::= WORDGLUE | LETTER { AlphaNum } | SYMBOL { SYMBOL }
Word      ::= WordPart1 { WORDGLUE WordPart }
```

That is, a `WordPart` is a string of alphanumerics or a string of symbols. A `Word` is a string of `WordPart`s separated by glue, and a `Word` cannot start with a `DIGIT`.

For example:

1. single words with no glue: x, $x1$, $\&+=$, λS, ΔS, $\exists \oplus$,
2. single words with glue: $\mathbb{P}\_x$, $\mathbb{P}_x$, x_+_y, $\_1$, ${}_1X$, $\exists_x$, $x_:e$, $x\_:\_e$
3. separate words: $\mathbb{P}\, x$, $x+y$, $\exists\, x$, $x:e$

Words can contain subscript digits (since the subcript markers are glue); there is a potential ambiguity in *trailing* subscript digits. This is because decoration strokes are the stroke characters ′ ? !, and also the subscript digits (comprising a begin subscript marker, a digit, and an end subscript marker). Is a trailing subscript digit part of the Word, or a stroke? The *Standard* Z lexis requires trailing subscript digits to be lexed as strokes, not part of the word. For example:

1. x_1 is the word x and the stroke $_1$
2. x_{a0} is the word x_a and the stroke $_0$
3. x_{0a} is the word x_{0a}
4. x_{0a12} is the word x_{0a}, the stroke $_1$, and the stroke $_2$
5. x^{b_0} is the word x^{b_0}

Occasionally, this flexibility with subscripts leads to 'surprises', requiring additional white space to separate letters. For example, $x_1 : X$ is lexed as the two words $x_1 :$ and X, because the subscript 1 has a preceding 'begin subscript', and a following 'end subscript' that glues it to the colon symbol; there is no glue between the colon and the X.

In the following toolkit, we make use of this lexis to define templates for cartesian square $X^{2\times}$ (§9), relation iteration r^n (§31), stream displays $_n\langle a, b, c\rangle$ (§34), etc.

10.4 Newlines and Line breaking

A newline character might be either a *soft newline* (white space) or a *hard newline* (low precedence predicate conjunction). Its context determines which it is.

It must be a soft newline if something is needed before or after it to complete a phrase:

- before or after an infix token
- after a prefix token or opening bracket
- before a postfix token or closing bracket

Elsewhere it is a hard newline (nothing is needed before or after it).

For example:

1. soft: p `NL` $\wedge\ q$, $a +$ `NL` b, seq `NL` X, a `NL` $^\sim$, $a($ `NL` $b)$, $[b$ `NL` $]$
2. hard: a `NL` (b), *true* `NL` *false*

Part III

Sets

Chapter 11

Sets, Types and Values

11.1 Introduction

Z is a language of predicates about typed sets. Although it can express ideas of great subtlety, the basic concepts are very simple.

11.2 Z types

Each set in Z may contain elements only of a single homogeneous type. Types are built up from basic types (*given sets*) using the power set constructor (the set of all subsets) or one of the forms of product: Cartesian product and schema type.

Sets can also be defined generically, for example, $X \leftrightarrow Y$, the set of all relations between X and Y. Consider the case of the natural numbers $\mathbb{N}$, the real numbers $\mathbb{R}$, and the Z numerical type $\mathbb{A}$, with $\mathbb{N} \subseteq \mathbb{R} \subseteq \mathbb{A}$. When a generic set is used, for example, as $\mathbb{R} \leftrightarrow \mathbb{N}$, its formal generic parameters are instantiated with some actual sets, here $\mathbb{R}$ and $\mathbb{N}$, whose types, here both $\mathbb{A}$, are used to find the actual type of the instantiated generic set, here $\mathbb{P}(\mathbb{A} \times \mathbb{A})$.

Generic sets can also be defined to accept their parameters in square brackets, for example $\varnothing[\mathbb{N} \times \mathbb{P}\,\mathbb{Z}]$. With these the parameter may instead be supplied implicitly, with no actual set being provided, provided that the type can be inferred from the context. In such a case the actual set is assumed to be the largest possible set of that type. For example, in the predicate $\varnothing \subseteq \mathbb{N}$, the empty set is assumed to have a generic parameter equal to the largest set that makes this predicate type consistent, namely $\mathbb{A}$.

Types are statically determinable: it is possible to write a type-checker that can unfailingly and in a finite time check whether type constraints have been observed, and if so decide the types of all expressions, including the value of all implicit

instantiations.

11.3 New sets

There are several ways to introduce sets into Z specifications. The simplest is to declare a given set. Such a declaration implies nothing about the set except that it exists, though we can add predicates that tell us more about it. Alternatively we can introduce a set using a 'free type' declaration, which is equivalent to a given set declaration together with predicates that constrain its structure quite tightly.

We can construct new sets and elements from existing sets and elements in a variety of ways, using for example:

- type constructors, to build new sets directly from existing sets: power set $\mathbb{P}\, X$ (sets of sets), and products $X \times Y$ (sets of tuples) and $[\ x : X;\ y : Y\ ]$ (sets of bindings)
- finite displays, to write down the members of a set explicitly: set display $\{x, y, z\}$, sequence display $\langle x, y, z\rangle$
- comprehensions, to define the members of a set implicitly, as those elements satisfying some condition: set comprehensions $\{\ x, y : X \mid x \neq y \bullet (x, y, x)\ \}$, λ-expressions $\lambda\, x : \mathbb{N} \bullet x + 1$
- schema calculus, to form new schemas (sets of bindings) from old
- selection, to extract part of a product: from a tuple $(x, y, z).2$ or from a binding $S.x$
- function application, where the supplied argument must occur precisely once in the domain of the set of pairs modelling the function, and the result is the corresponding range element. There are several ways of writing function application, but if f denotes the function and x the argument, then $f\, x$, adding parentheses if necessary, is always one of them.

11.4 Set membership

Sets are subject to the *axiom of extension*: 'two sets are equal if and only if they have the same members'. So sets are characterised fully by what members they have; if two sets have the same members, they are the same set. For any set a the totality of decisions for each candidate member (of the right type), as to whether it is a member of a, completely encapsulates the identity of a.

11.5 Structure of the Part

In §12 we describe some useful simple operations. In §13 we define operations for combining sets. In §14 we define the subset relations on sets. In §15 we define finiteness.

Chapter 12

Simple operations

Here we provide some initial simple operators.

- underlying type signature, σ
- boolean expressions, $\mathbb{B}$, $\top$, $\bot$
- negated core relation predicates, $\_ \notin \_$, $\_ \neq \_$
- Cartesian square, $\_^{2\times}$

Type signature

Intent

Obtain the underlying type of a set.

Definition

$$\sigma[X] == \lambda\, a : \mathbb{P}\, X \bullet X$$

Examples

- using σ with a specific generic instantiation results in the instantiating set: $\sigma[\mathbb{Z}]\mathbb{N} = \mathbb{Z}$ and $\sigma[[\ x : \mathbb{Z}\ ]][\ x : \mathbb{N}\ ] = [\ x : \mathbb{Z}\ ]$.
- using σ with an implicit generic instantiation results in the underlying type: $\sigma\mathbb{N} = \mathbb{A}$ and $\sigma[\ x : \mathbb{N}\ ] = [\ x : \mathbb{A}\ ]$.
- A schema written with its declaration equal to its signature is *normalised*: $S = [\ \sigma S \mid S\ ]$

Boolean expressions

Intent

Provide a type that can be used for Boolean-valued variables.

Definitions

The empty schema, and its schema negation, are schema expressions that act like Boolean variables when used as predicates.

$$\top == [\ \mid true\ ]$$
$$\bot == [\ \mid false\ ]$$
$$\mathbb{B} == \{\top, \bot\}$$

Motivation

If we wish to model the decision as to whether an item has a particular property, the usual Z style is to define the set of all the items that have the property, and then to ask whether the item is a member of that set. For example, to determine if a number n is prime, we ask if $n \in$ prime.

Because of this style, Boolean variables (named for the English logician, George Boole, 1815–1864, who is regarded as the founder of modern symbolic logic) 'true' and 'false' are not usually used in Z. (See the Modelling membership or flags choice pattern and the *Boolean flag* antipattern in [Stepney *et al.* 2003].) We use membership of the set of values having a particular attribute, rather than having a function that yields the attribute as a Boolean value (for example, finite $prime = false$), as many other notations would do.

However, Z can also be used circumstances other than pure specification. For example, consider the translation of another notation to Z, where that other notation does not share Z's syntactic distinction between predicates and expressions. Wherever a predicate p is used as a Boolean-valued expression in that language, empty schemas enable the straightforward translation to $[\ \mid p\ ]$ in Z. Another example is refinement toward code, where the code uses a Boolean data type.

The definition of *Boolean* given here is superior to the occasionally used free type $Boolean ::= False \mid True$, because the definition of its corresponding values true and false as schemas allows their use as predicates.

Laws

■ **Law 12.1** When used as predicates, $\top$ and $\bot$ act as *true* and *false*.

$$\vdash \top$$
$$\vdash \neg \bot$$

This law justifies the claim that these expressions can be used as Boolean-valued variables. So all the predicate laws given in §6 are also valid laws about schema expressions used as predicates when *true* and *false* are replaced with $\top$ and $\bot$ respectively, the schema-constrained generic are replaced by Boolean-valued variables, and the logical operators are interpreted as schema calculus operators.

For example, the predicate law of the excluded middle can be written using schema-constrained generic expressions as

$$[\dagger P] \vdash P \lor \neg P$$

(see §5, Generic conjectures : variant on schema-constrained generics), or using explicit Boolean-valued variables as

$$p : \mathbb{B} \vdash p \lor \neg p$$

•

Negated core relation predicates

Intent

Putting a slash through an infix relation to denote its negation is a common mathematical convention: $a \not R\ b \Leftrightarrow \neg\ a\ R\ b$. Z's operator template scheme is not powerful enough to define a general such 'slash' operator. So we define explicit relations for inequality and non-membership, since these arise very commonly in Z specifications. (The relations $=$ and $\in$ are part of the core language.) Other negated relations could be defined in a similar manner if they were found to be useful.

Definition

non-membership:

relation $(\_ \notin \_)$

$$\_ \notin \_\ [X] == \{\ x : X;\ a : \mathbb{P}\, X \mid \neg\ x \in a\ \}$$

inequality:

relation $(\_ \neq \_)$
$\_ \neq \_\ [X] == \{\ x, y : X \mid \neg\ x = y\ \}$

Examples

1. $1 \neq 2$
2. $\neg\ a \neq a$
3. $a \neq b \Rightarrow b \neq a$
4. $a \neq b \Rightarrow a \notin \{b\}$

Laws

Law 12.2 Inequality is irreflexive (§24) and symmetric (§24)

$[X]\ \vdash (\_ \neq \_)[X] \in \text{irreflexive}\ X \cap \text{symmetric}\ X$

Law 12.3 Inequality and the identity relation (§24) partition the Cartesian square (§12) of their parameter set.

$[X]\ \vdash \langle (\_ \neq \_)[X], \text{id}\ X \rangle\ \text{partition}\ X^{2\times}$
$[X]\ x, y : X \vdash x = y \vee x \neq y$

---•

Cartesian square

Intent

The Cartesian square of a set is the Cartesian product of that set with itself.

Definition

function $(\_^{2\times})$
$\_^{2\times}\ [X] == \lambda\ a : \mathbb{P}\ X \bullet a \times a$

In graph terms, every element is connected to every other element by an arc.

Examples

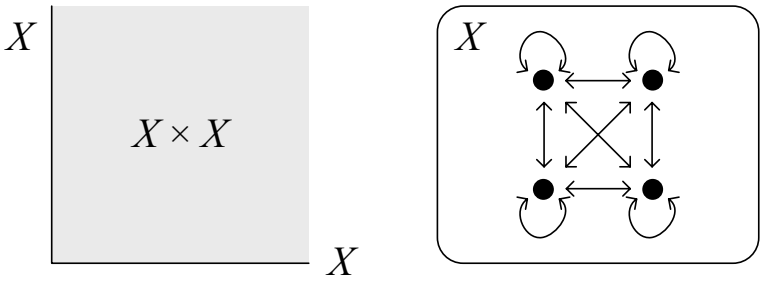


Figure 12.1 Cartesian square example

1. Figure 12.1 shows an example of a Cartesian square. (For an explanation of our diagramming conventions used here, see appendix A).
2. $\{x\}^{2\times} = \{(x, x)\}$
3. $\{x, y\}^{2\times} = \{(x, x), (x, y), (y, x), (y, y)\}$

Laws

Law 12.4 Cartesian square is a total injection.

$$[X] \vdash X^{2\times} \in X \rightarrowtail X$$

Law 12.5 Cartesian square is a reflexive, symmetric, transitive relation.

$$[X] \vdash X^{2\times} \in \text{reflexive}\, X \cap \text{symmetric}\, X \cap \text{transitive}\, X$$

Chapter 13

Basic set operations

In this chapter we define relations and functions that apply to sets of any matching types.

- empty set: $\varnothing$
- union: $(\_ \cup \_)$, $\bigcup$
- intersection: $(\_ \cap \_)$, $\bigcap$
- difference: $(\_ \setminus \_)$
- symmetric difference: $(\_ \ominus \_)$
- distribution properties
- closure property

As we noted in §12, rather than using a Boolean-valued function to determine if an item has a particular property, we can use a set to capture all such items. Continuing that approach, many of the set operations described in this chapter are analogues of the core language constructs for combining predicates; for example, set union — items that have this property *or* that property — is the analogue of logical disjunction: if the set p captures those items that have property P, and the set q captures those with property Q, then $p \cup q$ captures those with property $P \vee Q$. (The particular analogy is often clear from the set operation's definition.) Many of the set laws thus follow directly from the analogous predicate laws.

When we specify a set, we can do so by specifying the properties that all its elements have. Often it is clearer to specify a quite complicated property in stages. First specify simple properties, and then combine the simple sets in some manner to form a set with the desired property. Two sets can be combined to give a third in different ways:

- Union: the result has all the elements of each; 'all these properties, or all those properties, or both'.

- Intersection: the result has only the common elements of each; 'all these properties, and also all those properties'.
- Difference: the result has all the elements of one that are not in the other; 'all these properties, but not those properties'.
- Symmetric difference: the result has all the elements of each that are not in the other 'all these properties, or all those properties, but not both'.

Empty Set

Intent

The simplest set is the empty set, the set with no elements. Z is a typed language, so the empty set is generic in its type.

Definition

Empty set:

$$\varnothing[X] == \{\ x : X \mid \mathit{false}\ \}$$

The empty set is the set analogue of the predicate *false*; the empty set captures those items that have the property *false*.

Laws

Law 13.1 No element is a member of the empty set. A set is not empty precisely when it has at least one element.

$$[X]\ x : X \vdash x \notin \varnothing$$
$$[X]\ \vdash X \neq \varnothing \Leftrightarrow (\ \exists x : X \bullet \mathit{true}\ )$$

Law 13.2 An empty set is a subset of every set of that type. The only subset of the empty set is itself.

$$[X]\ \vdash \varnothing \subseteq X$$
$$[X]\ \mid X \subseteq \varnothing \vdash X = \varnothing$$

•

Set union

Intent

The union of two sets is a set that has all the elements of both. The generalised union of an arbitrary number of sets is a set that has all the elements of all the sets.

Definition

Set union:

$$\begin{array}{l}\text{function } 30 \text{ leftassoc } (\_ \cup \_) \\ \_ \cup \_ \, [X] == \lambda\, a, b : \mathbb{P}\, X \bullet \{\ x : X \mid x \in a \lor x \in b\ \}\end{array}$$

The set formed from the union of a and b has precisely those elements that are elements of a or of b or of both. Set union is the set analogue of logical disjunction: if the set p captures those items that have property P, and the set q captures those with property Q, then $p \cup q$ captures those with property $P \lor Q$.

Generalised set union:

$$\bigcup [X] == \lambda\, \alpha : \mathbb{P}\, \mathbb{P}\, X \bullet \{\ x : X \mid (\ \exists\, a : \alpha \bullet x \in a\ )\ \}$$

The set formed from the generalised union of a set of sets has precisely those elements that are elements of at least one of the constituent sets. Generalised set union is the set analogue of existential quantification.

We can define a *generalised* union this way, on a set of sets, precisely because set union is associative, commutative and idempotent (see the section on laws below). Later we define ways of distributing a binary operation over a set when it does not have all these properties (when is not idempotent, so that repeated elements need to be considered, in §22; when additionally it is not commutative, so that the order of application needs to be considered, in §38).

Examples

1. Figure 13.1 shows examples of set union.
2. $\vdash \{1, 3\} \cup \{2, 3\} = \{1, 2, 3\}$

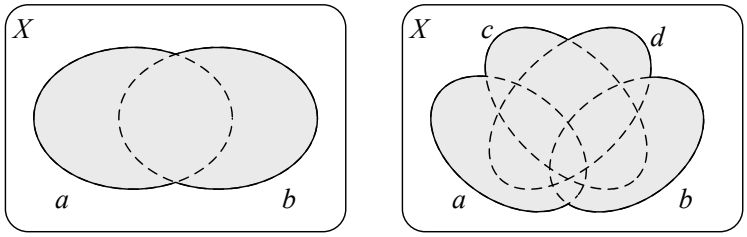


Figure 13.1 Venn diagrams of union $a \cup b$ or $\bigcup\{a, b\}$, and of union $a \cup b \cup c \cup d$ or $\bigcup\{a, b, c, d\}$.

3. The domain of the exponential function is defined as the union of three parts
   $\vdash \operatorname{dom}(\_ ** \_) = (\{0\} \times \mathbb{R}_+) \cup (\mathbb{R}_\pm \times \mathbb{Z}) \cup (\mathbb{R}_+ \times \mathbb{R})$
4. We can 'update' a relation $r : X \leftrightarrow Y$ (§16) by adding a new pair
   $r' = r \cup \{x \mapsto y\}$
5. $[X]\ a : \mathbb{P}\, X \vdash \bigcup\{a\} = a$
6. $[X]\ a, b : \mathbb{P}\, X \vdash \bigcup\{a, b\} = a \cup b$
7. If we have a function that maps labels to sets of things, $f : L \nrightarrow \mathbb{P}\, X$, we can require that the particular sets together exhaust the set X by imposing the condition
   $\bigcup(\operatorname{ran} f) = X$

Laws

Law 13.3 Set union and generalised union are total surjections (§21).

$$[X]\ \vdash (\_ \cup \_)[X] \in (\mathbb{P}\, X)^{2\times} \twoheadrightarrow \mathbb{P}\, X$$
$$[X]\ \vdash \bigcup[X] \in \mathbb{P}\,\mathbb{P}\, X \twoheadrightarrow \mathbb{P}\, X$$

Law 13.4 Set union is closed, commutative and associative, and has the empty set as an identity element. Hence it forms an abelian monoid (§23) over sets. Similarly for finite sets (§15).

$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{AbelianMonoid}[\mathbb{P}\, X] \bullet g = \mathbb{P}\, a \wedge (\_ \diamond \_) = (\_ \cup \_) \wedge e = \varnothing$$
$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{AbelianMonoid}[\mathbb{P}\, X] \bullet g = \mathbb{F}\, a \wedge (\_ \diamond \_) = (\_ \cup \_) \wedge e = \varnothing$$

Law 13.5 As a consequence of associativity and idempotence (law 13.46), set union and generalised union combine thus:

$$[X]\ \alpha : \mathbb{P}\,\mathbb{P}\,X;\ b : \mathbb{P}\,X \vdash \bigcup \alpha \cup b = \bigcup\{\ a : \alpha \bullet a \cup b\ \}$$

■ **Law 13.6** Generalised union of a set comprehension can be reduced to a set comprehension.

$$[X, Y]\ \alpha : \mathbb{P}\,\mathbb{P}\,X;\ f : \mathbb{P}\,X \to \mathbb{P}\,Y \vdash$$
$$\bigcup\{\ a : \alpha \bullet f\ a\ \} = \{\ y : Y \mid (\ \exists\, a : \alpha \bullet y \in f\ a\ )\ \}$$

■ **Law 13.7** Nested generalised unions can be flattened.

$$[X, Y, Z]\ \alpha : \mathbb{P}\,\mathbb{P}\,X;\ \beta : \mathbb{P}\,\mathbb{P}\,Y;\ f : \mathbb{P}\,X \times \mathbb{P}\,Y \to \mathbb{P}\,Z \vdash$$
$$\bigcup\{\ a : \alpha \bullet \bigcup\{\ b : \beta \bullet f(a, b)\ \}\ \} = \bigcup\{\ a : \alpha;\ b : \beta \bullet f(a, b)\ \}$$

Law 13.8 Every element of a generalised union is disjoint from some set b precisely when the union itself is disjoint from b.

$$[X]\ \alpha : \mathbb{P}\,\mathbb{P}\,X;\ b : \mathbb{P}\,X \vdash (\ \forall\, a : \alpha \bullet a \cap b = \varnothing\ ) \Leftrightarrow \bigcup \alpha \cap b = \varnothing$$

■ **Law 13.9** Intersection distributes through set union.

$$[X]\ \alpha : \mathbb{P}\,\mathbb{P}\,X;\ \beta : \mathbb{P}\,\mathbb{P}\,X \vdash \bigcup \alpha \cap \bigcup \beta = \bigcup\{\ a : \alpha;\ b : \beta \bullet a \cap b\ \}$$
$$[X]\ a, a', b : \mathbb{P}\,X \vdash (a \cup a') \cap b = (a \cap b) \cup (a' \cap b)$$

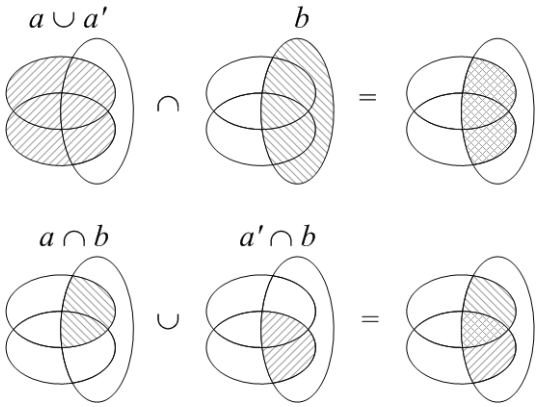


Figure 13.2 illustration of law 13.9: intersection distributes through set union

Law 13.10 Law 13.9 and law 14.11 give us that set union is subset order-preserving.

$$[X]\ a, a', b : \mathbb{P}\, X \mid a \subseteq a' \vdash a \cup b \subseteq a' \cup b$$

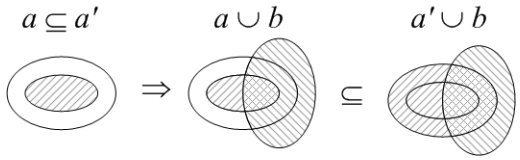


Figure 13.3 illustration of law 13.10: union is subset order preserving

■ **Law 13.11** Generalised union distributes through union.

$$[X]\ A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\, X \vdash \bigcup(\bigcup(\!|\ A\ |\!)) = \bigcup(\bigcup A)$$
$$[X]\ \alpha, \beta : \mathbb{P}\,\mathbb{P}\, X \vdash \bigcup \alpha \cup \bigcup \beta = \bigcup(\alpha \cup \beta)$$

So when using successive generalised unions to flatten sets of sets it does not change the result if the order of flattening is from the outside inwards, or from the inside outwards.

For example:

$$\begin{array}{l} A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,\mathbb{Z} \mid A = \{\{prime, even\}, \{odd\}\} \vdash \\ \quad \bigcup(\bigcup A) \\ \qquad = \bigcup(\bigcup\{\{prime, even\}, \{odd\}\}) \\ \qquad = \bigcup(\{prime, even\} \cup \{odd\}) \\ \qquad = \bigcup\{prime, even, odd\} \\ \qquad = prime \cup even \cup odd \\ \quad \wedge \bigcup(\bigcup(\!|\ A\ |\!)) \\ \qquad = \bigcup(\bigcup(\!|\ \{\{prime, even\}, \{odd\}\}\ |\!)) \\ \qquad = \bigcup\{\bigcup\{prime, even\}, \bigcup\{odd\}\} \\ \qquad = \bigcup\{prime \cup even, odd\} \\ \qquad = prime \cup even \cup odd \end{array}$$

Law 13.12 Law 13.11 and law 14.11 give us that generalised union is subset order-preserving (§26).

$$[X]\ \alpha, \beta : \mathbb{P}\,\mathbb{P}\, X \mid \alpha \subseteq \beta \vdash \bigcup \alpha \subseteq \bigcup \beta$$

Law 13.13 Specialising law 14.12 to $f = \bigcup$ gives

$$[X]\ A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,X \vdash \bigcup(\bigcap A) \subseteq \bigcap(\bigcup(\!|\ A\ |\!))$$
$$[X]\ \alpha, \beta : \mathbb{P}\,\mathbb{P}\,X \vdash \bigcup(\alpha \cap \beta) \subseteq \bigcup\alpha \cap \bigcup\beta$$

■ **Law 13.14** Generalised union distributes through intersection when all distinct elements are pairwise disjoint.

$$[X]\ A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,X \mid (\ \forall\, a, b : \bigcup A \mid a \neq b \bullet a \cap b = \varnothing\ ) \vdash \bigcup(\bigcap A) = \bigcap(\bigcup(\!|\ A\ |\!))$$
$$[X]\ \alpha, \beta : \mathbb{P}\,\mathbb{P}\,X \mid (\ \forall\, a, b : \alpha \cup \beta \mid a \neq b \bullet a \cap b = \varnothing\ ) \vdash \bigcup(\alpha \cap \beta) = \bigcup\alpha \cap \bigcup\beta$$

Generalised union does not distribute through intersection in general.

For example, consider

$$[X]\ x, y, z : X;\ A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,X \mid A = \{\{\{x\}, \{x, y\}\},\ \{\{x\}, \{y\}, \{x, z\}\}\} \vdash \bigcup(\bigcap A) = \bigcup\{\{x\}\} = \{x\} \land \bigcap(\bigcup(\!|\ A\ |\!)) = \bigcap\{\{x, y\},\ \{x, y, z\}\} = \{x, y\}$$

Law 13.15 Generalised union of the power set restores the original set.

$$[X]\ \vdash \bigcup(\mathbb{P}\,X) = X$$

Law 13.16 If α is a set of sets, then it is contained in the power set of its own generalised union.

$$[X]\ \alpha : \mathbb{P}\,\mathbb{P}\,X \vdash \alpha \subseteq \mathbb{P}(\bigcup\alpha)$$

The previous two laws bring out the way generalised union is in some sense the inverse of power set.

Law 13.17 Generalised union of the empty set of sets is empty.

$$[X]\ \vdash \bigcup \varnothing[\mathbb{P}\,X] = \varnothing[X]$$

The generalised union of the empty set of sets can be explained as follows: generalised union is defined in terms of an existential quantification, which is *false* over an empty set ('there is a pink unicorn' is *false*), so there are no elements satisfying the condition, and $\bigcup \varnothing = \varnothing$. Contrast this with generalised intersection (law 13.26).

Law 13.18 A union is empty precisely when each set in the union is empty.

$$[X]\ \alpha : \mathbb{P}\,\mathbb{P}\,X \vdash \bigcup \alpha = \varnothing \Leftrightarrow (\ \forall\, a : \alpha \bullet a = \varnothing\ )$$

———————————•

Set intersection

Intent

The intersection of two sets is a set that has the elements common to both. The generalised intersection of an arbitrary number of sets is a set that has the elements common to all.

Definition

Set intersection:

$$\begin{array}{l}\text{function } 40\ \text{leftassoc}\ (\_ \cap \_)\\ \_ \cap \_\ [X] == \lambda\, a, b : \mathbb{P}\,X \bullet \{\ x : X \mid x \in a \land x \in b\ \}\end{array}$$

The set formed from the intersection of a and b has precisely those elements that are elements of a and of b. Set intersection is the set analogue of logical conjunction; if the set p captures those items that have property P, and the set q captures those with property Q, then $p \cap q$ captures those with property $P \land Q$.

Notice that the precedence of $\cap$ is higher than that of $\cup$, following the fact that the precedence of $\land$ is higher than that of $\lor$.

Generalised set intersection:

$$\bigcap[X] == \lambda\, \alpha : \mathbb{P}\,\mathbb{P}\,X \bullet \{\ x : X \mid (\ \forall\, a : \alpha \bullet x \in a\ )\ \}$$

The set formed from the generalised intersection of a set of sets has precisely those elements that are elements of every one of the constituent sets. Generalised set intersection is the set analogue of universal quantification.

Examples

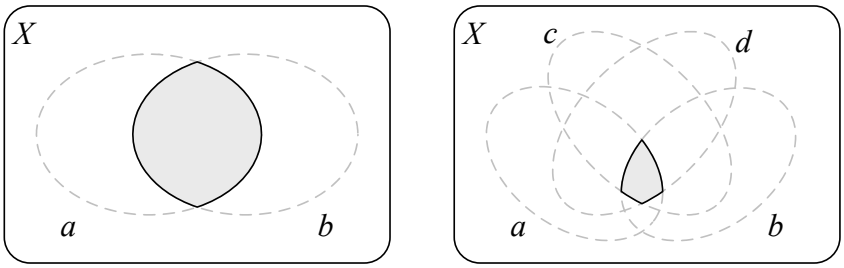


Figure 13.4 Venn diagrams of intersection $a \cap b$ or $\bigcap\{a, b\}$, and of intersection $a \cap b \cap c \cap d$ or $\bigcap\{a, b, c, d\}$.

1. Figure 13.4 shows examples of set intersection.
2. $\vdash \{1, 3\} \cap \{2, 3\} = \{3\}$
3. $[X]\ a, b : \mathbb{P}\, X \vdash \bigcap\{a, b\} = a \cap b$
4. $\vdash \bigcap\{\{2, 3, 4\}, \{1, 3, 5\}, \{2, 3\}\} = \{3\}$
5. $[X]\ x, y : X \vdash \bigcap\{\{x, y\}\} = \{x, y\}$
6. We define a total injection (§21) to be a function that is both total and an injection
   $X \rightarrowtail Y == (X \to Y) \cap (X \nrightarrow Y)$
7. We define a finite sequence (§34) to be both a sequence and finite
   $\operatorname{seq} X == \operatorname{sequence} X \cap (\mathbb{N} \pfun X)$
8. We define the finite subsets of a set (§15) to be the smallest set that both contains $\varnothing$ and is closed under union with singletons
   $\mathbb{F}\, X == \bigcap\{\ a : \mathbb{P}\,\mathbb{P}\, X \mid \varnothing \in a \land (\ \forall\, b : a;\ x : X \bullet b \cup \{x\} \in a\ )\ \}$
9. We define the transitive closure of a relation (§24) to be the smallest transitive relation containing r
   $[X]\ r : X \leftrightarrow X \vdash r^+ = \bigcap\{\ t : \operatorname{transitive} X \mid r \subseteq t\ \}$

Laws

Law 13.19 Set intersection and generalised intersection are total surjections (§21).

$[X]\ \vdash (\_ \cap \_)[X] \in (\mathbb{P}\, X)^{2\times} \twoheadrightarrow \mathbb{P}\, X$
$[X]\ \vdash \bigcap[X] \in \mathbb{P}\,\mathbb{P}\, X \twoheadrightarrow \mathbb{P}\, X$

Law 13.20 Set intersection is closed, commutative and associative, and has the generic parameter set as its identity element. Hence it forms an abelian monoid (§23) over sets. Similarly for finite sets (§15).

$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{AbelianMonoid}[\mathbb{P}\, X] \bullet g = \mathbb{P}\, a \wedge (\_ \diamond \_) = (\_ \cap \_) \wedge e = a$$
$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{AbelianMonoid}[\mathbb{P}\, X] \bullet g = \mathbb{F}\, a \wedge (\_ \diamond \_) = (\_ \cap \_) \wedge e = a$$

Law 13.21 As consequence of associativity and idempotence (law 13.46), set intersection and generalised intersection combine thus:

$$[X]\ \alpha : \mathbb{P}\,\mathbb{P}\, X;\ b : \mathbb{P}\, X \vdash \bigcap \alpha \cap b = \bigcap \{\ a : \alpha \bullet a \cap b\ \}$$

■ **Law 13.22** Generalised intersection of a set comprehension can be reduced to a set comprehension.

$$[X, Y]\ \alpha : \mathbb{P}\,\mathbb{P}\, X;\ f : \mathbb{P}\, X \to \mathbb{P}\, Y \vdash$$
$$\quad \bigcap \{\ a : \alpha \bullet f\ a\ \} = \{\ y : Y \mid (\ \forall\, a : \alpha \bullet y \in f\ a\ )\ \}$$

■ **Law 13.23** Nested generalised intersections can be flattened.

$$[X, Y, Z]\ \alpha : \mathbb{P}\,\mathbb{P}\, X;\ \beta : \mathbb{P}\,\mathbb{P}\, Y;\ f : \mathbb{P}\, X \times \mathbb{P}\, Y \to \mathbb{P}\, Z \vdash$$
$$\quad \bigcap \{\ a : \alpha \bullet \bigcap \{\ b : \beta \bullet f(a, b)\ \}\ \} = \bigcap \{\ a : \alpha;\ b : \beta \bullet f(a, b)\ \}$$

■ **Law 13.24** Union distributes through set intersection.

$$[X]\ \alpha, \beta : \mathbb{P}\,\mathbb{P}\, X \vdash \bigcap \alpha \cup \bigcap \beta = \bigcap \{\ a : \alpha;\ b : \beta \bullet a \cup b\ \}$$
$$[X]\ a, a', b : \mathbb{P}\, X \vdash (a \cap a') \cup b = (a \cup b) \cap (a' \cup b)$$

Law 13.25 Hence law 14.11 gives us that set intersection is subset order-preserving.

$$[X]\ a, a', b : \mathbb{P}\, X \mid a \subseteq a' \vdash a \cap b \subseteq a' \cap b$$

Law 13.26 Generalised intersection of the empty set of sets is the whole parameter set of the union.

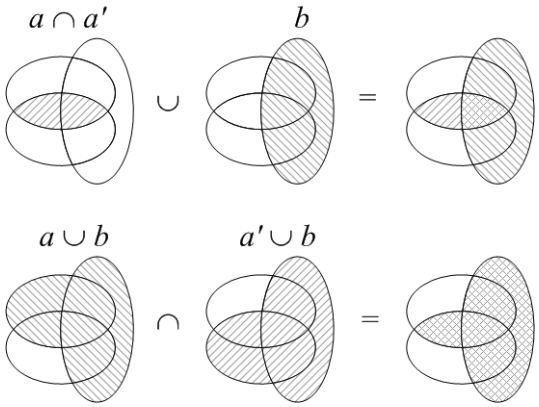


Figure 13.5 illustration of law 13.24: union distributes through set intersection

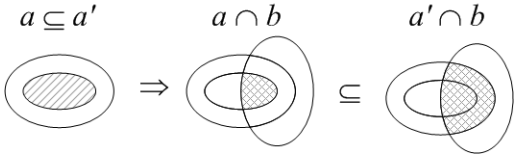


Figure 13.6 illustration of law 13.25: intersection is subset order preserving

$$[X] \ \vdash \bigcap [X]\varnothing = X$$

The generalised intersection of the empty set of sets can be explained as follows: generalised intersection is defined in terms of a universal quantification, which is *true* over an empty set ('all unicorns are pink' is *true*), so all elements of $\bigcap$'s parameter satisfy the condition, and so $\bigcap [X]\varnothing = X$. Contrast this with generalised union (law 13.17).

Law 13.27 If x is in a which is disjoint from b, then x is not in b.

$$[X] \ x : X;\ a, b : \mathbb{P}\, X \mid x \in a \wedge a \cap b = \varnothing \vdash x \notin b$$

■ **Law 13.28** Generalised intersection pseudo-distributes through union.

$$[X] \ A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\, X \vdash \bigcap(\bigcup A) = \bigcap(\bigcap(\!|\ A\ |\!))$$
$$[X] \ \alpha, \beta : \mathbb{P}\,\mathbb{P}\, X \vdash \bigcap(\alpha \cup \beta) = \bigcap \alpha \cap \bigcap \beta$$

For example:

$$[X]\ x, y, z : X;\ A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,X \mid A = \{\{\{x\}, \{x, y\}\},\ \{\{x, y, z\}, \{x, z\}\}\} \vdash \\ \bigcap(\bigcup A) = \bigcap\{\{x\}, \{x, y\}, \{x, y, z\}, \{x, z\}\} = \{x\} \\ \land \bigcap(\bigcap(\!|\ A\ |\!)) = \bigcap\{\{x\}, \{x, z\}\} = \{x\}$$

Law 13.29 Hence law 13.44 gives us that generalised intersection is subset order-reversing (§26).

$$[X]\ \alpha, \beta : \mathbb{P}\,\mathbb{P}\,X \mid \alpha \subseteq \beta \vdash \bigcap \beta \subseteq \bigcap \alpha$$

Law 13.30 Specialising law 14.15 to $f = \bigcap$ gives

$$[X]\ A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,X \vdash \bigcup(\bigcap(\!|\ A\ |\!)) \subseteq \bigcap(\bigcap A)$$
$$[X]\ \alpha, \beta : \mathbb{P}\,\mathbb{P}\,X \vdash \bigcap \alpha \cup \bigcap \beta \subseteq \bigcap(\alpha \cap \beta)$$

Generalised intersection does not in general pseudo-distribute through intersection.

For example, consider:

$$[X]\ x, y, z : X;\ A : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,X \mid A = \{\{\{x\}, \{x, y\}\},\ \{\{x, y\}, \{x, z\}\}\} \vdash \\ \bigcap(\bigcap A) = \bigcap\{\{x, y\}\} = \{x, y\} \\ \land \bigcup(\bigcap(\!|\ A\ |\!)) = \bigcup\{\{x\}\} = \{x\}$$

•

Set difference

Intent

The difference of two sets has just those elements that are in the first but not the second of the sets.

Definition

Set difference:

$$\text{function } 30\,\text{leftassoc}\,(\_ \setminus \_)$$
$$\_ \setminus \_\ [X] == \lambda\, a, b : \mathbb{P}\,X \bullet \{\ x : X \mid x \in a \land x \notin b\ \}$$

The difference of a and b is a set that has precisely those elements that are elements of a but not of b. If a is put equal to X's type, we specialise set difference to set

complement with respect to X's type. Such set complement is the set analogue of (unary) logical negation; if the set $\mathcal{X}$ captures all items of the correct type (all items that have property *true*), and the set q captures those with property Q, then $\mathcal{X} \setminus q$ captures those with property $\neg\, Q$.

Examples

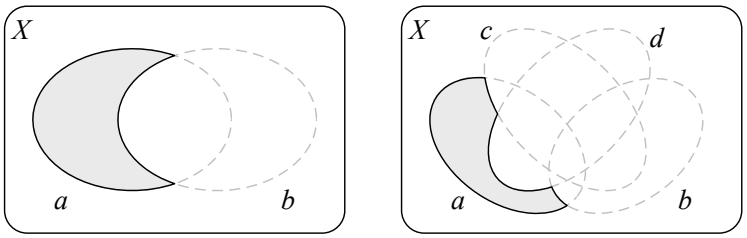


Figure 13.7 Venn diagrams of set difference $a \setminus b$, and of difference $a \setminus b \setminus c \setminus d$.

1. Figure 13.7 shows an example of set difference.
2. $\vdash \{1, 3\} \setminus \{2, 3\} = \{1\}$
3. $\vdash \{1, 3\} \setminus \{1, 3\} = \varnothing$
4. $\vdash \{1, 3\} \setminus \{2\} = \{1, 3\}$
5. If we have a set of all items of interest, *all*, and a distinguished subset of it, *special*, then the non-special items are
   $\mathit{ordinary} == \mathit{all} \setminus \mathit{special}$
6. The domain of the logarithmic function *log* (§33) includes $\mathbb{R}_+ \setminus \{1\}$
7. An odd number is an integer that is not even
   $\mathit{even} == \{\ i : \mathbb{Z} \bullet 2 * i\ \}$
   $\mathit{odd} == \mathbb{Z} \setminus \mathit{even}$
8. A prime number is a number that is not composite
   $\mathbb{N}_2 == \mathbb{N} \setminus \{0, 1\}$
   $\mathrm{prime} == \mathbb{N}_2 \setminus \mathrm{composite}$

Laws

Law 13.31 Set difference is a total surjection (§21).

$$[X] \ \vdash (\_ \setminus \_)[X] \in (\mathbb{P}\, X)^{2\times} \twoheadrightarrow \mathbb{P}\, X$$

Law 13.32 Two sets are disjoint precisely when one is a subset of the other's complement. Then set subtraction leaves the set unchanged.

$$[X]\ a, b : \mathbb{P}\, X \vdash a \cap b = \varnothing \Leftrightarrow a \subseteq X \setminus b$$
$$[X]\ a, b : \mathbb{P}\, X \vdash a \cap b = \varnothing \Leftrightarrow a \setminus b = a$$

Law 13.33 Set difference is not commutative.

$$[X]\ a, b : \mathbb{P}\, X \vdash a \setminus b = b \setminus a \Leftrightarrow a = b$$

Law 13.34 Set difference is not associative (figure 13.8).

$$[X]\ a, b, c : \mathbb{P}\, X \vdash a \setminus (b \setminus c) = (a \setminus b) \setminus c \Leftrightarrow a \cap c = \varnothing$$
$$[X]\ a, b, c : \mathbb{P}\, X \vdash a \setminus (b \setminus c) = (a \setminus b) \cup (a \cap c)$$
$$[X]\ a, b, c : \mathbb{P}\, X \vdash (a \setminus b) \setminus c = a \setminus (b \cup c) = (a \setminus b) \cap (a \setminus c)$$

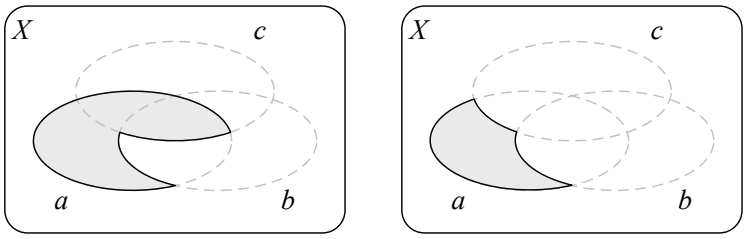


Figure 13.8 Illustration of law 13.34: set difference is not associative: $a \setminus (b \setminus c)$ and $(a \setminus b) \setminus c$

Law 13.35 For set difference, the empty set is a right identity and left zero.

$$[X]\ \vdash X \setminus X = \varnothing$$
$$[X]\ \vdash X \setminus \varnothing = X$$
$$[X]\ \vdash \varnothing \setminus X = \varnothing$$

Law 13.36 Set difference distributes through union on the left, and through intersection on the left when the collection of sets is non-empty.

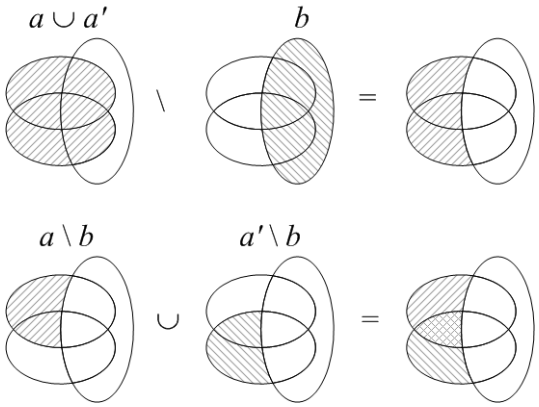


Figure 13.9 illustration of law 13.36a: set difference distributes through union on left

$$[X]\ \alpha : \mathbb{P}\,\mathbb{P}\,X;\ b : \mathbb{P}\,X \vdash \bigcup\{\ a : \alpha \bullet a \setminus b\ \} = \bigcup \alpha \setminus b$$
$$[X]\ a, a', b : \mathbb{P}\,X \vdash (a \setminus b) \cup (a' \setminus b) = (a \cup a') \setminus b$$

$$[X]\ \alpha : \mathbb{P}_1\,\mathbb{P}\,X;\ b : \mathbb{P}\,X \vdash \bigcap \alpha \setminus b = \bigcap\{\ a : \alpha \bullet a \setminus b\ \}$$
$$[X]\ a, a', b : \mathbb{P}\,X \vdash (a \cap a') \setminus b = (a \setminus b) \cap (a' \setminus b)$$

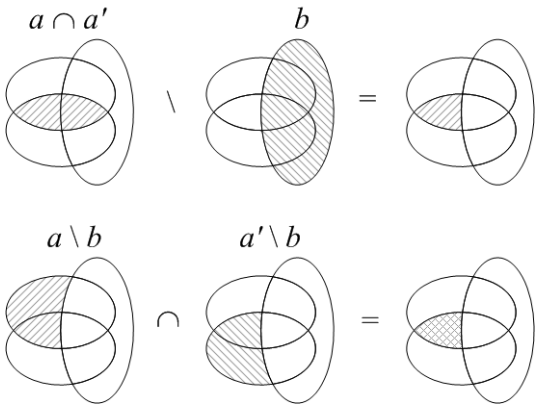


Figure 13.10 illustration of law 13.36b: set difference distributes through intersection on left

The intersection law does not hold in general for an empty collection of sets:

$$
\begin{array}{l}
[X]\ b : \mathbb{P}\, X \vdash \\
\quad \bigcap \varnothing \setminus b = X \setminus b \\
\quad \wedge \bigcap \{\ a : \varnothing[\mathbb{P}\, X] \bullet a \setminus b\ \} = \bigcap \varnothing = X
\end{array}
$$

Law 13.37 Set difference pseudo-distributes through union on the right, and through intersection on the right when the collection of sets is non-empty.

$$[X]\ a : \mathbb{P}\, X;\ \beta : \mathbb{P}\, \mathbb{P}\, X \vdash \bigcup \{\ b : \beta \bullet a \setminus b\ \} = a \setminus \bigcap \beta$$
$$[X]\ a, b, b' : \mathbb{P}\, X \vdash (a \setminus b) \cup (a \setminus b') = a \setminus (b \cap b')$$

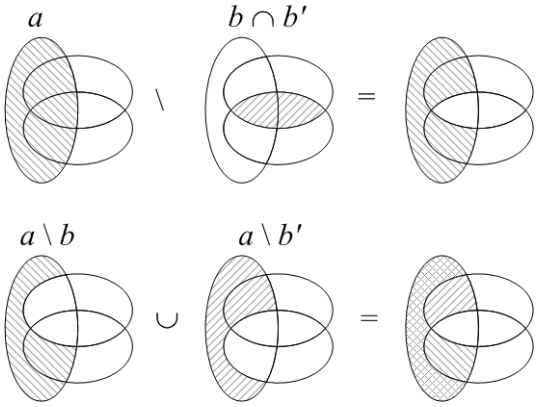


Figure 13.11 illustration of law 13.37a: set difference pseudo-distributes through union on right

$$[X]\ a : \mathbb{P}\, X;\ \beta : \mathbb{P}_1\, \mathbb{P}\, X \vdash a \setminus \bigcup \beta = \bigcap \{\ b : \beta \bullet a \setminus b\ \}$$
$$[X]\ a, b, b' : \mathbb{P}\, X \vdash a \setminus (b \cup b') = (a \setminus b) \cap (a \setminus b')$$

The intersection law does not hold in general for an empty collection of sets:

$$
\begin{array}{l}
[X]\ a : \mathbb{P}\, X \vdash \\
\quad a \setminus \bigcup \varnothing = a \setminus \varnothing = a \\
\quad \wedge \bigcap \{\ b : \varnothing[\mathbb{P}\, X] \bullet a \setminus b\ \} = \bigcap \varnothing = X
\end{array}
$$

If in the pseudo-distributive laws we put a equal to X's type, thereby specialising set difference to set complement with respect to X's type, we get de Morgan's laws.

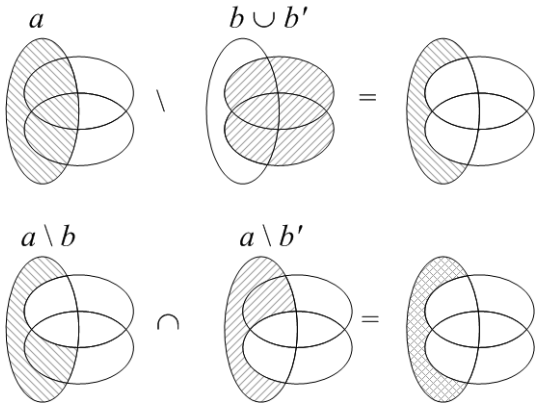


Figure 13.12 illustration of law law 13.37b: set difference pseudo-distributes through intersection on right

Law 13.38 The distributive law gives us that set difference is subset order-preserving on its first argument; the pseudo-distributive law gives us that set difference is subset order-reversing on its second argument.

$$[X]\ a, a', b : \mathbb{P}\, X \mid a \subseteq a' \vdash a \setminus b \subseteq a' \setminus b$$
$$[X]\ a, b, b' : \mathbb{P}\, X \mid b \subseteq b' \vdash a \setminus b' \subseteq a \setminus b$$

Law 13.39 Set difference has some more pseudo-distributive properties:

$$[X]\ a, b, c : \mathbb{P}\, X \vdash (a \setminus b) \cap c = (a \cap c) \setminus b = a \cap (c \setminus b)$$
$$[X]\ a, b, c : \mathbb{P}\, X \vdash (a \setminus b) \cup c = (a \cup c) \setminus (b \setminus c)$$

Symmetric set difference

Intent

The symmetric difference (sometimes called symmetric sum) of two sets has just those elements that are in precisely one of the sets.

Definition

Symmetric set difference:

$$\text{function } 30 \text{ leftassoc } (\_ \ominus \_)$$
$$\_ \ominus \_ \, [X] == \lambda\, a, b : \mathbb{P}\, X \bullet \{\ x : X \mid \neg\, (x \in a \Leftrightarrow x \in b)\ \}$$

The symmetric difference of a and b is a set that has precisely those elements that are elements of a or of b but not of both. Symmetric set difference is the set analogue of 'exclusive or' (which does not occur as a primitive construct in the core language); if the set p captures those items that have property P, and the set q captures those with property Q, then $p \ominus q$ captures those with property $\neg\, (P \Leftrightarrow Q)$.

Examples

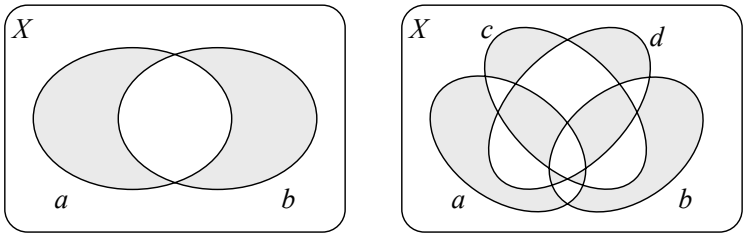


Figure 13.13 Venn diagrams of symmetric set difference, $a \ominus b$ and $a \ominus b \ominus c \ominus d$.

1. Figure 13.13 shows examples of symmetric set difference.
2. $\vdash \{1, 3\} \ominus \{2, 3\} = \{1, 2\}$
3. $\vdash \{1, 3\} \ominus \{1, 3\} = \varnothing$
4. $\vdash \{1, 3\} \ominus \{2\} = \{1, 2, 3\}$

Laws

Law 13.40 Symmetric set difference is a total surjection (§21).

$$[X] \ \vdash (\_ \ominus \_)[X] \in (\mathbb{P}\, X)^{2\times} \twoheadrightarrow \mathbb{P}\, X$$

Law 13.41 Symmetric set difference is the difference of the union and the intersection, and the union of the differences.

$$[X]\ a, b : \mathbb{P}\, X \vdash a \ominus b = (a \cup b) \setminus (a \cap b) = (a \setminus b) \cup (b \setminus a)$$

Law 13.42 Symmetric set difference is closed, commutative and associative, has the empty set as its identity element, and the identity function as its inverse. Hence it forms an abelian group (§23) over sets. Similarly for finite sets (§15).

$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{AbelianGroup}[\mathbb{P}\, X] \bullet \\ g = \mathbb{P}\, a \wedge (\_ \diamond \_) = (\_ \ominus \_) \wedge e = \varnothing \wedge inv = \mathrm{id}\, \mathbb{P}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{AbelianGroup}[\mathbb{P}\, X] \bullet \\ g = \mathbb{F}\, a \wedge (\_ \diamond \_) = (\_ \ominus \_) \wedge e = \varnothing \wedge inv = \mathrm{id}\, \mathbb{F}\, a$$

Law 13.43 Distributive and pseudo-distributive properties of symmetric set difference:

$$[X]\ a, a', b : \mathbb{P}\, X \vdash (a \ominus a') \cap b = (a \cap b) \ominus (a' \cap b)$$
$$[X]\ a, a', b : \mathbb{P}\, X \vdash (a \ominus a') \cup b = (a \cup b) \ominus (a' \setminus b)$$
$$[X]\ a, a', b : \mathbb{P}\, X \vdash b \setminus (a \ominus a') = (b \setminus a) \ominus (b \cap a')$$
$$[X]\ a, a', b : \mathbb{P}\, X \vdash (a \ominus a') \setminus b = (a \setminus b) \ominus (a' \setminus b)$$

—————————————•

Distribution properties

Intent

To show the interdependencies among the various set distribution laws.

Laws

■ **Law 13.44** If a total operation on sets distributes through set difference, then it distributes through intersection, union, and symmetric difference.

$$[X, Y]\ f : \mathbb{P}\, X \to \mathbb{P}\, Y;\ a, b : \mathbb{P}\, X \mid \forall\, a', b' : \mathbb{P}\, X \bullet f(a' \setminus b') = f\ a' \setminus f\ b' \vdash \\ f(a \cap b) = f\ a \cap f\ b \\ \wedge\, f(a \cup b) = f\ a \cup f\ b \\ \wedge\, f(a \ominus b) = f\ a \ominus f\ b$$

Law 13.45 If a function distributes through symmetric set difference and also through union or intersection, then it distributes through all four set operations.

$$[X, Y]\ f : \mathbb{P}\, X \to \mathbb{P}\, Y;\ a, b : \mathbb{P}\, X \mid \\ \forall\, a', b' : \mathbb{P}\, X \bullet f(a' \ominus b') = f\ a' \ominus f\ b' \land f(a' \cup b') = f\ a' \cup f\ b' \vdash \\ f(a \cap b) = f\ a \cap f\ b \\ \land\, f(a \setminus b) = f\ a \setminus f\ b$$

$$[X, Y]\ f : \mathbb{P}\, X \to \mathbb{P}\, Y;\ a, b : \mathbb{P}\, X \mid \\ \forall\, a', b' : \mathbb{P}\, X \bullet f(a' \ominus b') = f\ a' \ominus f\ b' \land f(a' \cap b') = f\ a' \cap f\ b' \vdash \\ f(a \cup b) = f\ a \cup f\ b \\ \land\, f(a \setminus b) = f\ a \setminus f\ b$$

Examples

1. an operation that distributes through $\ominus$ but not through the other set operations is given by the function f where
   $[X]\ c : \mathbb{F}_1\, X;\ f == \lambda\, a : \mathbb{P}\, X \bullet \{\ x : \mathbb{N} \mid x = \#(c \cap a) \bmod 2 = 1\ \}$
2. an operation that distributes through $\cup$ and $\cap$ but not through the other set operations is given by the function f where
   $[X]\ c : \mathbb{P}_1\, X;\ f == \lambda\, a : \mathbb{P}\, X \bullet c$
3. an operation that distributes through $\cap$ but not through the other set operations is given by $\mathbb{P}$.
4. an operation that distributes through $\cup$ but not through the other set operations is given by the function
   $f[X] == \lambda\, a : \mathbb{P}\, X \bullet \{b : \mathbb{P}\, X \mid a \cap b \neq \varnothing\}$
5. a total set to set operation that does not distribute through any of the four set operations is given by the function $\vec{\#}$ (§30).

---•

Closure property

Intent

If we start with any two sets of the same type, *any* expression involving those sets and any number of uses of the four operations $\cup, \cap, \setminus, \ominus$ can be simplified to an expression using at most one operation. (There is no corresponding law if we restrict ourselves to just the three operations $\cup$, $\cap$ and $\setminus$.)

Laws

Law 13.46 Given any two starting sets of the same type, we can generate a finite set of sets closed under the four operations $\cup, \cap, \setminus, \ominus$.

$$[X] \; a, b : \mathbb{P}\, X;\; \alpha : \mathbb{P}\mathbb{P}\, X \mid \alpha = \{\varnothing, a, b, a \cap b, a \cup b, a \setminus b, b \setminus a, a \ominus b\} \vdash \\ \forall\, c, d : \alpha \bullet \{c \cap d, c \cup d, c \setminus d, c \ominus d\} \subseteq \alpha$$

It is useful to present all the possibilities by using four 8×8 tables (one for each operator). The first argument is down the side, the second argument along the top. Three of the operators are commutative, so only half the table need be shown.

$\cup$	$\varnothing$	a	b	$a \cup b$	$a \cap b$	$a \setminus b$	$b \setminus a$	$a \ominus b$
$\varnothing$	$\varnothing$	a	b	$a \cup b$	$a \cap b$	$a \setminus b$	$b \setminus a$	$a \ominus b$
a	–	a	$a \cup b$	$a \cup b$	a	a	$a \cup b$	$a \cup b$
b	–	–	b	$a \cup b$	b	$a \cup b$	b	$a \cup b$
$a \cup b$	–	–	–	$a \cup b$	$a \cup b$	$a \cup b$	$a \cup b$	$a \cup b$
$a \cap b$	–	–	–	–	$a \cap b$	a	b	$a \cup b$
$a \setminus b$	–	–	–	–	–	$a \setminus b$	$a \ominus b$	$a \ominus b$
$b \setminus a$	–	–	–	–	–	–	$b \setminus a$	$a \ominus b$
$a \ominus b$	–	–	–	–	–	–	–	$a \ominus b$

$\cap$	$\varnothing$	a	b	$a \cup b$	$a \cap b$	$a \setminus b$	$b \setminus a$	$a \ominus b$
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
a	$-$	a	$a \cap b$	a	$a \cap b$	$a \setminus b$	$\varnothing$	$a \setminus b$
b	$-$	$-$	b	b	$a \cap b$	$\varnothing$	$b \setminus a$	$b \setminus a$
$a \cup b$	$-$	$-$	$-$	$a \cup b$	$a \cap b$	$a \setminus b$	$b \setminus a$	$a \ominus b$
$a \cap b$	$-$	$-$	$-$	$-$	$a \cap b$	$\varnothing$	$\varnothing$	$\varnothing$
$a \setminus b$	$-$	$-$	$-$	$-$	$-$	$a \setminus b$	$\varnothing$	$a \setminus b$
$b \setminus a$	$-$	$-$	$-$	$-$	$-$	$-$	$b \setminus a$	$b \setminus a$
$a \ominus b$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$a \ominus b$

$\setminus$	$\varnothing$	a	b	$a \cup b$	$a \cap b$	$a \setminus b$	$b \setminus a$	$a \ominus b$
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
a	a	$\varnothing$	$a \setminus b$	$\varnothing$	$a \setminus b$	$a \cap b$	a	$a \cap b$
b	b	$b \setminus a$	$\varnothing$	$\varnothing$	$b \setminus a$	b	$a \cap b$	$a \cap b$
$a \cup b$	$a \cup b$	$b \setminus a$	$a \setminus b$	$\varnothing$	$a \ominus b$	b	a	$a \cap b$
$a \cap b$	$a \cap b$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$a \cap b$	$a \cap b$	$a \cap b$
$a \setminus b$	$a \setminus b$	$\varnothing$	$a \setminus b$	$\varnothing$	$a \setminus b$	$\varnothing$	$a \setminus b$	$\varnothing$
$b \setminus a$	$b \setminus a$	$b \setminus a$	$\varnothing$	$\varnothing$	$b \setminus a$	$b \setminus a$	$\varnothing$	$\varnothing$
$a \ominus b$	$a \ominus b$	$b \setminus a$	$a \setminus b$	$\varnothing$	$a \ominus b$	$b \setminus a$	$a \setminus b$	$\varnothing$

$\ominus$	$\varnothing$	a	b	$a \cup b$	$a \cap b$	$a \setminus b$	$b \setminus a$	$a \ominus b$
$\varnothing$	$\varnothing$	a	b	$a \cup b$	$a \cap b$	$a \setminus b$	$b \setminus a$	$a \ominus b$
a	$-$	$\varnothing$	$a \ominus b$	$b \setminus a$	$a \setminus b$	$a \cap b$	$a \cup b$	b
b	$-$	$-$	$\varnothing$	$a \setminus b$	$b \setminus a$	$a \cup b$	$a \cap b$	a
$a \cup b$	$-$	$-$	$-$	$\varnothing$	$a \ominus b$	b	a	$a \cap b$
$a \cap b$	$-$	$-$	$-$	$-$	$\varnothing$	a	b	$a \cup b$
$a \setminus b$	$-$	$-$	$-$	$-$	$-$	$\varnothing$	$a \ominus b$	$b \setminus a$
$b \setminus a$	$-$	$-$	$-$	$-$	$-$	$-$	$\varnothing$	$a \setminus b$
$a \ominus b$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$\varnothing$

We can use these tables to read off many properties of the appropriate set operations. For example, we see that set union and intersection are *idempotent* (since we can read off that $a \cup a = a$ and $a \cap a = a$), but that set difference and symmetric set difference are not.

---•

Chapter 14

Subsets

In this chapter we define relations and functions that apply to sets of any matching types.

- subset: $(\_ \subseteq \_)$
- proper subset: $(\_ \subset \_)$
- non-empty subsets: $\mathbb{P}_1\, X$
- distributive laws

In following chapters we go on to define relations and functions on sets of pairs and other more complex structures.

Subset

Intent

The subset relations tell us when one set is wholly contained in another.

Definition

Subset:

$$\begin{array}{l} \text{relation}\,(\_ \subseteq \_) \\ \_ \subseteq \_\,[X] == \{\ a, b : \mathbb{P}\, X \mid a \in \mathbb{P}\, b\ \} \end{array}$$

a is a subset of b precisely when it is a member of the power set of b. The subset relation is the set analogue of logical implication; if the set p captures those items that have property P, and the set q captures those with property Q, then $p \subseteq q$ expresses the property $P \Rightarrow Q$.

Examples

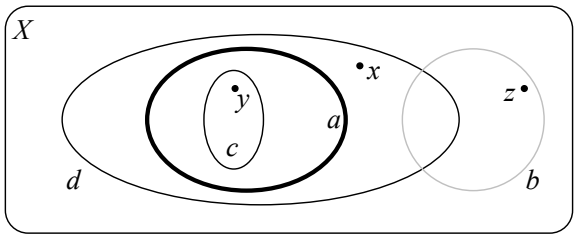


Figure 14.1 Venn diagram of subset relations: $c \subseteq a$, $\neg\ c \subseteq b$, $c \subset d$

1. Figure 14.1 shows examples of the subset relation.
2. $\vdash \{1\} \subseteq \{1,3\} \subseteq \{1,2,3\}$
3. $\vdash$ prime $\subseteq \mathbb{N}$
4. $\vdash \neg\ \{2\} \subseteq \{1,3\}$
5. $\vdash \neg\ \{1,3\} \subseteq \{2,3\}$
6. $\vdash \neg$ prime $\subseteq$ *odd*

Laws

Law 14.1 a is a subset of b precisely when every element of a is also an element of b.

$$[X]\ a, b : \mathbb{P}\, X \vdash a \subseteq b \Leftrightarrow (\ \forall\, x : a \bullet x \in b\ )$$

Law 14.2 A set is a subset precisely when it is an identity for union. A set is a subset precisely when it is a fixed point for intersection. A set is a subset precisely when the difference is the empty set.

$$[X]\ a, b : \mathbb{P}\, X \vdash a \subseteq b \Leftrightarrow a \cup b = b$$
$$[X]\ a, b : \mathbb{P}\, X \vdash a \subseteq b \Leftrightarrow a \cap b = a$$
$$[X]\ a, b : \mathbb{P}\, X \vdash a \subseteq b \Leftrightarrow a \setminus b = \varnothing$$

Law 14.3 Subset is reflexive (every set is a subset of itself, $a \subseteq a$), antisymmetric (two distinct sets cannot each be a subset of the other, $a \subseteq b \land b \subseteq a \Leftrightarrow a = b$), and transitive ($a \subseteq b \subseteq c \Rightarrow a \subseteq c$). Hence the subset relation forms an order on sets (§26).

$$[X] \ \vdash (\_ \subseteq \_)[X] \in \mathrm{reflexiveOrder}\,\mathbb{P}\,X$$

Law 14.4 Subset has generalised intersection as its greatest lower bound (§26), and generalised union as its least upper bound.

$$[X] \ \vdash \mathrm{glb}(\_ \subseteq \_)[X] = \bigcap$$
$$[X] \ \vdash \mathrm{lub}(\_ \subseteq \_)[X] = \bigcup$$

Law 14.5 The law about the *lub* (law 26.28) can be specialised for the subset order; a set has each member of a collection of sets as a subset precisely when it has their union as a subset (figure 14.2).

$$[X] \ \alpha : \mathbb{P}\,\mathbb{P}\,X;\ b : \mathbb{P}\,X \vdash (\ \forall\, a : \alpha \bullet a \subseteq b\ ) \Leftrightarrow \bigcup \alpha \subseteq b$$
$$[X] \ a, a', b : \mathbb{P}\,X \vdash a \subseteq b \wedge a' \subseteq b \Leftrightarrow a \cup a' \subseteq b$$

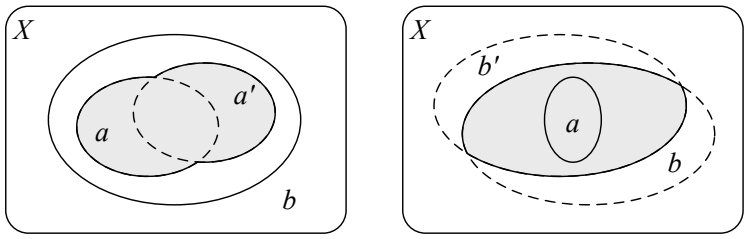


Figure 14.2 Illustration of the specialisation of the laws about *lub* and *glb* to the subset order. Law 14.5: $a \subseteq b \wedge a' \subseteq b \Leftrightarrow a \cup a' \subseteq b$ and law 14.6: $a \subseteq b \wedge a \subseteq b' \Leftrightarrow a \subseteq b \cap b'$

Law 14.6 The law about the *glb* (law 26.28) can be specialised for the subset order; a set is a subset of each member of a collection of sets precisely when it is a subset of their intersection (figure 14.2).

$$[X] \ a : \mathbb{P}\,X;\ \alpha : \mathbb{P}\,\mathbb{P}\,X \vdash (\ \forall\, b : \alpha \bullet a \subseteq b\ ) \Leftrightarrow a \subseteq \bigcap \alpha$$
$$[X] \ a, b, b' : \mathbb{P}\,X \vdash a \subseteq b \wedge a \subseteq b' \Leftrightarrow a \subseteq b \cap b'$$

———————————————•

Proper subset

Intent

The proper subset relations tell us when one set is wholly contained in another, and is not identical.

Definition

Proper subset:

$$\text{relation}\,(\_ \subset \_)$$
$$\_ \subset \_\,[X] == (\_ \subseteq \_)[X] \cap (\_ \neq \_)$$

a is a proper subset of b precisely when it is a subset of b and distinct from b; if every element of a is also a member of b and there is some member of b that is not a member of a.

Examples

1. $\#\{x, z\} = 2 \Rightarrow \{x\} \subset \{x, z\}$
2. $\varnothing \subset \{x, z\}$
3. $\text{prime} \subset \mathbb{N}$
4. $\neg\, \{x, z\} \subset \{x, z\}$

Laws

Law 14.7 a is a proper subset of b precisely when a is a subset of b, but b is not a subset of a

$$[X]\ a, b : \mathbb{P}\, X \vdash a \subset b \Leftrightarrow a \subseteq b \land \neg\, b \subseteq a$$
$$[X]\ a, b : \mathbb{P}\, X \vdash a \subset b \Leftrightarrow a \subseteq b \land (\ \exists\, x : b \bullet x \notin a\ )$$

Law 14.8 Proper subset is irreflexive (no set is a proper subset of itself), anti-symmetric (two distinct sets cannot each be a proper subset of the other), and transitive ($a \subset b \subset c \Rightarrow a \subset c$). Hence the proper subset relation forms an order on sets (§26).

$$[X] \vdash (\_ \subset \_)[X] \in \mathrm{irreflexiveOrder}\,\mathbb{P}\,X$$

Law 14.9 An empty set is a proper subset of precisely the non-empty sets of its type.

$$[X] \vdash \varnothing \subset X \Leftrightarrow X \neq \varnothing$$

•

Non-empty subsets

Intent

When we declare $a : \mathbb{P}\,X$, we are saying that the set a is some subset of X. The case where we also want to have a non-empty, $a \neq \varnothing$, is sufficiently common that there is special notation for it, $a : \mathbb{P}_1\,X$.

Discussion

There are many cases where we wish to exclude the empty set from consideration (for example, non-empty finite sets, non-empty sequences). We would like to be able to define a general mechanism for doing this.

If we were to define a postfix operator, to mimic the conventional use of notation such as $\mathrm{seq}_1\,X$, we would have to write $(\mathrm{seq}\,X)_1$. A prefix notation would allow us to write ${}_1\,\mathrm{seq}\,X$, associating the 'non-empty' symbol more closely with the relevant operator. However, neither of these is close enough to the familiar conventional notation, so we choose instead to define explicitly $name_1 == name \setminus \{\varnothing\}$ for each name where we wanta non-empty variant.

Even if we were to define a general non-emptiness operator, non-empty subsets $\mathbb{P}_1$ could not be defined using it, because $\mathbb{P}$ is a core-language element, not a constructed element: $\mathbb{P}_1$ is a single name; ${}_1\,\mathbb{P}$ would be syntactically incorrect.

Definition

Non-empty subsets:

$$\mathbb{P}_1\,X == \mathbb{P}\,X \setminus \{\varnothing\}$$

Examples

1. $\mathbb{P}_1 \varnothing[X] = \varnothing$
2. $\mathbb{P}_1\{x\} = \{\{x\}\}$
3. $\mathbb{P}_1\{x, y\} = \{\{x\}, \{y\}, \{x, y\}\}$
4. $\mathbb{P}_1\{x, y, z\} = \{\{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}$
5. If we want to state in a predicate that a is a non-empty set of elements drawn from X, we can say either $\varnothing \neq a \subseteq X$ or $a \in \mathbb{P}_1 X$.

Laws

Law 14.10 $\mathbb{P}_1 X$ is empty precisely when X is empty; $\mathbb{P}_1 X$ contains X precisely when X is not empty.

$$[X] \vdash X = \varnothing \Leftrightarrow \mathbb{P}_1 X = \varnothing$$
$$[X] \vdash X \neq \varnothing \Leftrightarrow X \in \mathbb{P}_1 X$$

•

Distribution property

Laws

Law 14.11 A function that distributes through union or intersection is subset-order preserving.

$$[X, Y]\ a, b : \mathbb{P} X;\ f : \mathbb{P} X \to \mathbb{P} Y \mid$$
$$(\ \forall a', b' : \mathbb{P} X \bullet f\ a' \cap f\ b' = f(a' \cap b')\ )$$
$$\vee\ (\ \forall a', b' : \mathbb{P} X \bullet f\ a' \cup f\ b' = f(a' \cup b')\ ) \vdash$$
$$a \subseteq b \Leftrightarrow f\ a \subseteq f\ b$$

Law 14.12 A subset-order-preserving map applied to unions and intersections of sets gives wider bounds than unions and intersections of the image of the set through the map (law 26.36).

$$[X, Y]\ \alpha : \mathbb{P}\mathbb{P} X;\ f : \mathbb{P} X \to \mathbb{P} Y \mid (\ \forall a, b : \mathbb{P} X \mid a \subseteq b \bullet f\ a \subseteq f\ b\ ) \vdash$$
$$\bigcup(f(\!|\ \alpha\ |\!)) \subseteq f(\bigcup \alpha)$$
$$\wedge\ f(\bigcap \alpha) \subseteq \bigcap(f(\!|\ \alpha\ |\!))$$

$$[X, Y]\ a, b : \mathbb{P}\, X;\ f : \mathbb{P}\, X \to \mathbb{P}\, Y \mid (\ \forall\, a', b' : \mathbb{P}\, X \mid a' \subseteq b' \bullet f\ a' \subseteq f\ b'\ ) \vdash$$
$$f\ a \cup f\ b \subseteq f(a \cup b)$$
$$\wedge\ f(a \cap b) \subseteq f\ a \cap f\ b$$

Law 14.13 As a direct consequence of the previous two laws, if a function distributes through set union, then there is a corresponding subset law for intersection; if a function distributes through set intersection, then there is a corresponding subset law for union.

$$[X, Y]\ f : \mathbb{P}\, X \to \mathbb{P}\, Y;\ \alpha : \mathbb{P}\,\mathbb{P}\, X \mid \forall\, \alpha' : \mathbb{P}\,\mathbb{P}\, X \bullet f(\bigcup \alpha') = \bigcup(f(\!|\ \alpha'\ |\!)) \vdash$$
$$f(\bigcap \alpha) \subseteq \bigcap(f(\!|\ \alpha\ |\!))$$

$$[X, Y]\ f : \mathbb{P}\, X \to \mathbb{P}\, Y;\ a, b : \mathbb{P}\, X \mid \forall\, a', b' : \mathbb{P}\, X \bullet f(a' \cup b') = f\ a' \cup f\ b' \vdash$$
$$f(a \cap b) \subseteq f\ a \cap f\ b$$

$$[X, Y]\ f : \mathbb{P}\, X \to \mathbb{P}\, Y;\ \alpha : \mathbb{P}\,\mathbb{P}\, X \mid \forall\, \alpha' : \mathbb{P}\,\mathbb{P}\, X \bullet f(\bigcap \alpha') = \bigcap(f(\!|\ \alpha'\ |\!)) \vdash$$
$$\bigcup(f(\!|\ \alpha\ |\!)) \subseteq f(\bigcup \alpha)$$

$$[X, Y]\ f : \mathbb{P}\, X \to \mathbb{P}\, Y;\ a, b : \mathbb{P}\, X \mid \forall\, a', b' : \mathbb{P}\, X \bullet f(a' \cap b') = f\ a' \cap f\ b' \vdash$$
$$f\ a \cup f\ b \subseteq f(a \cup b)$$

Law 14.14 A function that pseudo-distributes through union or intersection is subset order-reversing.

$$[X, Y]\ a, b : \mathbb{P}\, X;\ f : \mathbb{P}\, X \to \mathbb{P}\, Y \mid$$
$$(\ \forall\, a', b' : \mathbb{P}\, X \bullet f\ a' \cap f\ b' = f(a' \cup b')\ )$$
$$\vee\ (\ \forall\, a', b' : \mathbb{P}\, X \bullet f\ a' \cup f\ b' = f(a' \cap b')\ ) \vdash$$
$$a \subseteq b \Leftrightarrow f\ b \subseteq f\ a$$

Law 14.15 A subset-order-reversing map applied to unions and intersections of sets gives wider bounds than intersections and unions of the image of the set through the map (law 26.39).

$$[X, Y]\ \alpha : \mathbb{P}\,\mathbb{P}\, X;\ f : \mathbb{P}\, X \to \mathbb{P}\, Y \mid (\ \forall\, a, b : \mathbb{P}\, X \mid a \subseteq b \bullet f\ b \subseteq f\ a\ ) \vdash$$
$$f(\bigcup \alpha) \subseteq \bigcap(f(\!|\ \alpha\ |\!))$$
$$\wedge\ \bigcup(f(\!|\ \alpha\ |\!)) \subseteq f(\bigcap \alpha)$$

$$[X, Y]\ a, b : \mathbb{P}\, X;\ f : \mathbb{P}\, X \to \mathbb{P}\, Y \mid (\ \forall\, a', b' : \mathbb{P}\, X \mid a' \subseteq b' \bullet f\ b' \subseteq f\ a'\ ) \vdash$$
$$f(a \cup b) \subseteq f\ a \cap f\ b$$
$$\wedge\ f\ a \cup f\ b \subseteq f(a \cap b)$$

———————————————•

Chapter 15

Finiteness

Often in Z work, and particularly in general algebraic developments like those in this catalogue, we neither know nor need to know very much about the sets we use. The best practice is to make the minimum of assumptions, as this assists clarity and allows the greatest possibility of reuse of the structures we develop. In general a set may be finite, which we know if we have equated it to a finite display, or it may be countably infinite, like $\mathbb{N}$, the set of natural numbers, or it may be infinite and uncountable, like $\mathbb{P}\,\mathbb{N}$ or $\mathbb{R}$. To say that a set is uncountable means that it is not possible to make a list of its members, not even an infinite list. Particular sets of interest are usually finitely describable, but this does not make them finite or countable as sets.

In certain special cases, however, it may be useful, or necessary, to require a set to be finite. For example, a recursive definition that consists of a base case, plus a case that splits a set into two smaller ones, may be well-founded only when the set is finite, so that splitting it in two eventually reduces it to the base case.

- finite subsets: $\mathbb{F}\,X$
- non-empty finite subsets: $\mathbb{F}_1\,X$
- finiteness: (finite _)

Finite sets

Intent

Require a set to be finite.

Definitions

Finite subsets:

$$\begin{array}{l}\text{generic}\,(\mathbb{F}\_)\\ \mathbb{F}\,X == \bigcap\{\ a : \mathbb{P}\,\mathbb{P}\,X \mid \varnothing \in a \land (\ \forall\, b : a;\ x : X \bullet b \cup \{x\} \in a\ )\ \}\end{array}$$

The set of finite subsets is the smallest set that both contains the empty set and is closed under union with singleton sets.

Consider the following. Let us work within an (infinite) generic set X. Let a be the set of finite subsets of X. We know that $\varnothing$ is finite, so we have $\varnothing \in a$. We can be sure that if b is a finite set, and we take the union with a new (or existing) element, the result will be a finite set, so we have $\forall\, b : a;\ x : X \bullet b \cup \{x\} \in a$. Finally we assert that every finite set can be produced in this way, so that we can take the generalised intersection of all sets that have these two properties, and this defines $\mathbb{F}\,X$.

Non-empty finite subsets:

$$\mathbb{F}_1\,X == \mathbb{F}\,X \setminus \{\varnothing\}$$

The set of non-empty finite subsets excludes the empty set.

Finiteness predicate:

$$\begin{array}{l}\text{relation}\,(\text{finite}\_)\\ \text{finite}\_\,[X] == \mathbb{F}\,X\end{array}$$

For any set a the predicate *finite* a is true precisely when a is finite.

Examples

1. the empty set is finite: $[X] \vdash \text{finite}\,\varnothing[X]$
2. any singleton set is finite: $[X]\ x : X \vdash \text{finite}\{x\}$
3. $\vdash \neg\ \text{finite}\,\mathbb{N}_1$

Laws

Law 15.1 *induction principle*: if a property of a set holds for $\varnothing$, and is preserved through union with single elements, it holds for all finite sets. This principle follows directly from the definition. It can be used in the proofs of most of the laws below.

$$[X]\ a : \mathbb{F}\, X \vdash \forall\, A : \mathbb{P}\,\mathbb{P}\, a \mid \varnothing \in A \wedge (\ \forall\, \alpha : A;\ x : a \bullet \alpha \cup \{a\} \in A\ ) \bullet a \in A$$

Law 15.2 The intersection of a finite set with any set is finite (see also law 30.21).

$$[X]\ a, b : \mathbb{P}\, X \vdash \mathbb{F}(a \cap b) = \mathbb{F}\, a \cap \mathbb{P}\, b$$
$$[X]\ a : \mathbb{P}\, X;\ b : \mathbb{F}\, X \vdash \mathrm{finite}(a \cap b)$$

Law 15.3 The union of a finite collection of finite sets is finite.

$$[X]\ \alpha : \mathbb{F}\,\mathbb{F}\, X \vdash \mathrm{finite}(\bigcup \alpha)$$

Law 15.4 Subtracting a finite set from an infinite set leaves an infinite set.

$$[X]\ a : \mathbb{P}\, X;\ b : \mathbb{F}\, X \mid \neg\ \mathrm{finite}\, a \vdash \neg\ \mathrm{finite}(a \setminus b)$$
$$[X]\ a : \mathbb{P}\, X;\ b : \mathbb{F}\, X \mid \mathrm{finite}(a \setminus b) \vdash \mathrm{finite}\, a$$

Law 15.5 Any subset of a finite set is finite (see also law 30.22).

$$[X]\ a : \mathbb{P}\, X;\ b : \mathbb{F}\, X \mid a \subseteq b \vdash \mathrm{finite}\, a$$

Law 15.6 If a set is finite, there is no surjection (§21) onto it from a proper subset of itself. With a finite set, a proper subset is always smaller.

$$[X]\ a : \mathbb{P}\, X;\ b : \mathbb{F}\, X \mid a \subset b \vdash a \nrightarrow\!\!\!\!\rightarrow b = \varnothing$$

A surjection onto an infinite set can have a domain that is a proper subset of the range.

> For example, the function $\mathit{halve} : \mathit{even} \rightarrow\!\!\!\!\rightarrow \mathbb{N}$, that integer-divides its even argument by 2, has $\mathrm{ran}\, \mathit{halve} = \mathbb{N}$ and $\mathrm{dom}\, \mathit{halve} = \mathit{even} \subset \mathbb{N}$.

Law 15.7 *Pigeonhole principle*: if a set is finite, all its total homogeneous injections (§21) are necessarily surjections, and vice versa. Equivalently, if a set is finite, the range of all total homogeneous injections is the same as the domain.

$$[X]\ \mid \mathrm{finite}\, X \vdash X \rightarrowtail X = X \rightarrow\!\!\!\!\rightarrow X$$
$$[X]\ f : X \rightarrowtail X \mid \mathrm{finite}\, X \vdash \mathrm{ran}\, f = X$$

The fact that, for a finite set, a total injection onto itself is necessarily a surjection is used in some theorems in number theory.

A total homogenous injection on an infinite set can have a range that is a proper subset of the domain.

> For example, the function $double : \mathbb{N} \rightarrowtail \mathbb{N}$, that multiplies its argument by 2, has $\operatorname{dom} double = \mathbb{N}$ and $\operatorname{ran} double = even \subset \mathbb{N}$.

Law 15.8 A set is finite precisely when there is a surjection onto it from another finite set.

$$[X, Y] \vdash \mathbb{F}\, X = \{\ b : \mathbb{P}\, X \mid \exists\, c : \mathbb{F}\, Y \bullet c \twoheadrightarrow b \neq \varnothing\ \}$$

Law 15.9 A set is finite precisely when there is a bijection (§21) to it from an initial segment of the natural numbers (§29).

$$[X] \vdash \mathbb{F}\, X = \{\ b : \mathbb{P}\, X \mid \exists\, n : \mathbb{N} \bullet 1\,..\,n \rightarrowtail b \neq \varnothing\ \}$$

The n in the bijection above is the size of the set. There may be a bijection from *all* the natural numbers to an infinite set, in which case the infinite set is *countable*.

> For example, the prime numbers, the integers $\mathbb{Z}$, and the rationals $\mathbb{Q}$, are countable sets.

There may be no such bijection, in which case the infinite set is *uncountable*.

> For example, the power set of the natural numbers $\mathbb{P}\,\mathbb{N}$, and the reals $\mathbb{R}$, are uncountable sets.

Part IV

Binary relations

Chapter 16

Relations

A *relation* encapsulates the information as to whether two or more objects, taken together, have a particular property. The convention in Z, when describing which elements have some property, is to define the set of all elements that have the property. Similarly, the usual model of a relation is the set of ordered tuples that have the property of interest. Although one can work with tuples of any size, a common way in Z is to build up whatever is required in terms of *binary* relations, where the tuples are then ordered pairs.

- first and second components: first, second
- relation: $X \leftrightarrow Y$
- finite relations: $X \nrightarrow Y$
- maplet: $(\_ \mapsto \_)$
- inverse: $(\_^\sim)$
- dual laws

Such relations are just *sets* of pairs, so all the set operations introduced in the previous part are also applicable.

Tuple component selection

Intent

We can access the components of a tuple using a Z core language construct:

$$p = (x, y);\ p.1 = x;\ p.2 = y;\ p = (p.1, p.2)$$
$$q = (x, y, z);\ q.1 = x;\ q.2 = y;\ q.3 = z;\ q = (q.1, q.2, q.3)$$

For the common case of selection of components of a pair, we define *first* and *second*, which allow selection to be applied and composed.

Definition

first and second components:

$$\text{first}[X, Y] == \lambda\, p : X \times Y \bullet p.1$$
$$\text{second}[X, Y] == \lambda\, p : X \times Y \bullet p.2$$

Examples

1. the image of a relation through *first* is its domain
   $\text{first}(\!|\ r\ |\!) = \text{dom}\, r$
2. If we have a relation to pairs of items, $r : Z \leftrightarrow X \times Y$, then we can split it to form relations to the first of the pair and to the second of the pair
   $r = \{z_1 \mapsto (x_1, y_1), \ldots, z_n \mapsto (x_n, y_n)\}$
   $r \fatsemi \text{first} = \{z_1 \mapsto x_1, \ldots, z_n \mapsto x_n\}$
   $r \fatsemi \text{second} = \{z_1 \mapsto y_1, \ldots, z_n \mapsto y_n\}$

Laws

Law 16.1 *first* and *second* are total surjections (§21).

$$[X, Y] \vdash \text{first}[X, Y] \in X \times Y \twoheadrightarrow X$$
$$[X, Y] \vdash \text{second}[X, Y] \in X \times Y \twoheadrightarrow Y$$

Law 16.2 Any pair is fully characterised by its first and second components.

$$[X, Y]\ p : X \times Y \vdash p = (\text{first}\, p, \text{second}\, p)$$

•

Relation notation

Intent

Sets of pairs used as relations are so commonly used in Z specifications that there is a special infix notation for them.

Definition

relation:

$$\text{generic } 5 \text{ rightassoc } (\_ \leftrightarrow \_)$$
$$X \leftrightarrow Y == \mathbb{P}(X \times Y)$$

Examples

1. $\{(x_1, y_1), (x_1, y_2), (x_3, y_2), (x_5, y_6)\} \in X \leftrightarrow Y$
2. the full relation: $X \times Y \in X \leftrightarrow Y$
3. the empty relation: $\varnothing[X \times Y] \in X \leftrightarrow Y$
4. that the relation constructor distributes through non-empty generalised intersection, but not through the other set operators, can be readily deduced from the distributive properties of power set and Cartesian product.

$$[X, Y]\ a, a' : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash (a \cap a') \leftrightarrow b = a \leftrightarrow b \cap a' \leftrightarrow b$$

Finite relations

Intent

Define explicitly finite relations.

Definition

finite relations:

$$\text{generic } 5 \text{ rightassoc } (\_ \nrightarrow \_)$$
$$X \nrightarrow Y == \mathbb{F}(X \times Y)$$

Laws

Law 16.3 A relation is finite precisely when its domain and range are finite.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r \in X \nrightarrow Y \Leftrightarrow \text{finite}\, \text{dom}\, r \land \text{finite}\, \text{ran}\, r$$

Maplet notation

Intent

The maplet notation is syntactic sugar for a particular element of a relation, a pair. It can be used as emphasis, to draw attention to the fact that a particular pair is being considered to be an element of a relation, and to reduce cluttering parentheses.

Definition

maplet:

$$\begin{array}{l}\text{function } 10 \text{ leftassoc } (\_ \mapsto \_) \\ \_ \mapsto \_ \; [X, Y] == \lambda\, x : X;\; y : Y \bullet (x, y)\end{array}$$

Examples

1. $x \mapsto y = (x, y)$
2. $\{x_1 \mapsto y_1, x_1 \mapsto y_2, x_3 \mapsto y_2, x_5 \mapsto y_6\} = \{(x_1, y_1), (x_1, y_2), (x_3, y_2), (x_5, y_6)\}$

•

Relational inverse

Intent

Relational inverse swaps the components of the pairs.

Definition

inverse:

$$\begin{array}{l}\text{function } (\_^{\sim}) \\ \_^{\sim}\; [X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \{\; p : r \bullet p.2 \mapsto p.1 \;\}\end{array}$$

Examples

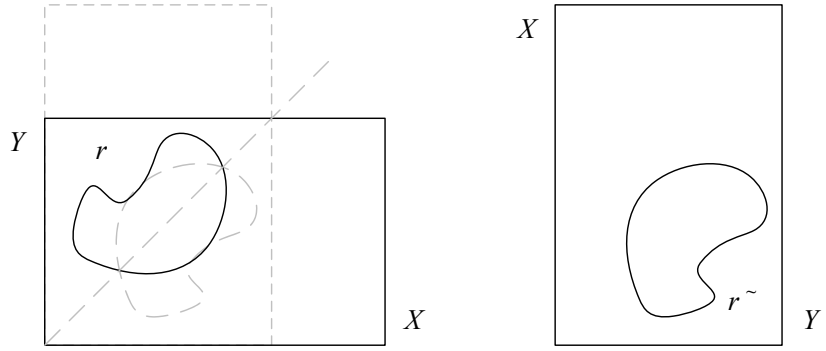


Figure 16.1 An example of relational inverse.

1. Figure 16.1 shows an example of relational inverse.
2. $\{x \mapsto y\}^\sim = \{y \mapsto x\}$
3. $\{x_1 \mapsto y_1, x_1 \mapsto y_2, x_3 \mapsto y_2, x_5 \mapsto y_6\}^\sim$
   $= \{y_1 \mapsto x_1, y_2 \mapsto x_1, y_2 \mapsto x_3, y_6 \mapsto x_5\}$

Laws

Law 16.4 Relational inverse is a bijection (§21).

$$[X, Y] \vdash (\_^\sim)[X, Y] \in (X \leftrightarrow Y) \rightarrowtail\!\!\!\!\!\rightarrow (Y \leftrightarrow X)$$

Law 16.5 Inverting twice restores the original relation.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r^{\sim\sim} = r$$

■ **Law 16.6** The complement of the inverse is the inverse of the complement.

$$[X, Y]\ r : X \leftrightarrow Y \vdash (Y \times X) \setminus r^\sim = ((X \times Y) \setminus r)^\sim$$

■ **Law 16.7** Relational inverse distributes through generalised union and generalised intersection.

$$[X, Y]\ \rho : \mathbb{P}(X \leftrightarrow Y) \vdash \bigcup\{\ r : \rho \bullet r^\sim\ \} = (\bigcup \rho)^\sim$$
$$[X, Y]\ r, s : X \leftrightarrow Y \vdash r^\sim \cup s^\sim = (r \cup s)^\sim$$
$$[X, Y]\ \rho : \mathbb{P}(X \leftrightarrow Y) \vdash (\bigcap \rho)^\sim = \bigcap\{\ r : \rho \bullet r^\sim\ \}$$
$$[X, Y]\ r, s : X \leftrightarrow Y \vdash (r \cap s)^\sim = r^\sim \cap s^\sim$$

■ **Law 16.8** Relational inverse distributes through set diffference.

$$[X, Y]\ r, s : X \leftrightarrow Y \vdash (r \setminus s)^\sim = r^\sim \setminus s^\sim$$

Law 16.9 As a corollary of laws 16.8 and 16.7, with law 13.45, inverse distributes through symmetric set difference.

$$[X, Y]\ r, s : X \leftrightarrow Y \vdash (r \ominus s)^\sim = r^\sim \ominus s^\sim$$

Law 16.10 As a corollary of law 26.37 specialised to subsets, along with law 16.8, relational inverse is subset order-preserving.

$$[X, Y]\ r, s : X \leftrightarrow Y \mid r \subseteq s \vdash r^\sim \subseteq s^\sim$$

———————————•

Deriving dual laws

Intent

Many of the operators defined below occur in dual pairs, one about the source set, and the other about the target set, and a law relating them using relational inverse. In such a case it is possible to derive a set of laws about the target set from those about the source set and the inverse law; hence only source set laws are given.

Derivation

For example, assume we have a pair of dual operators, source operator f_s and target operator f_t defined thus:

$[X, Y]$

$$f_s : (X \leftrightarrow Y) \times \mathbb{P}\, X \to \mathbb{P}\, Y$$
$$f_t : (X \leftrightarrow Y) \times \mathbb{P}\, Y \to \mathbb{P}\, X$$

$\ldots$

Assume that these operators are related by relational inverse as

$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash \exists\, \mathcal{G} : \mathbb{P}\, X \nrightarrow \mathbb{P}\, Y \bullet f_s(r^{\sim}, a) = \mathcal{G}(f_t(r, a))$$

Then any law about f_s such as

$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash \mathcal{P}(f_s(r, a), r, a)$$

can be translated into a dual law about f_t as follows. The names in the above law are arbitrary, so we can rewrite by swapping the names X as Y, and renaming r as $r^{\sim}$, a as b.

$$[X, Y]\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash \mathcal{P}(f_s(r^{\sim}, b), r^{\sim}, b)$$

Finally, use the inverse law linking the operators to substitute for (some) occurrences of $r^{\sim}$ to get the dual form of the law about f_t.

$$[X, Y]\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash \mathcal{P}(\mathcal{G}(f_t(r, b)), r^{\sim}, b)$$

———————————————•

Chapter 17

Domain and Range of relations

Domain and range

Intent

The domain of a relation is all those elements of the source set participating in the relation. Similarly the range is all those elements in the relation's target set.

Definition

domain and range:

$$\mathrm{dom}[X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \{\ p : r \bullet p.1\ \}$$
$$\mathrm{ran}[X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \{\ p : r \bullet p.2\ \}$$

The domain and range functions extract the set of first elements of all the pairs, and the set of second elements, respectively.

Examples

1. Figure 17.1 shows an example of domain and range.
2. $\mathrm{dom}\{x_1 \mapsto y_1, x_1 \mapsto y_2, x_3 \mapsto y_2, x_5 \mapsto y_6\} = \{x_1, x_3, x_5\}$
   $\mathrm{ran}\{x_1 \mapsto y_1, x_1 \mapsto y_2, x_3 \mapsto y_2, x_5 \mapsto y_6\} = \{y_1, y_2, y_6\}$
3. The domain of a sequence (§34) is a prefix of the positive numbers
   $\mathrm{dom}\langle a, b, c, d\rangle = 1\,..\,4$

Laws

Law 17.1 Range laws can be derived as 'duals' of domain laws, by using

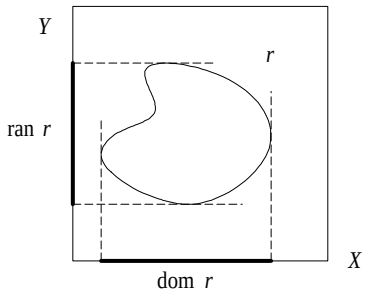


Figure 17.1 An example of the domain and range of a relation.

$$[X, Y]\ r : X \leftrightarrow Y \vdash \mathrm{dom}(r^{\sim}) = \mathrm{ran}\, r$$

Law 17.2 *dom* and *ran* are total surjections (§21).

$$[X, Y]\ \vdash \mathrm{dom}[X, Y] \in (X \leftrightarrow Y) \twoheadrightarrow \mathbb{P}\, X$$
$$[X, Y]\ \vdash \mathrm{ran}[X, Y] \in (X \leftrightarrow Y) \twoheadrightarrow \mathbb{P}\, Y$$

■ **Law 17.3** *dom* and *ran* distribute through generalised union.

$$[X, Y]\ \rho : \mathbb{P}(X \leftrightarrow Y) \vdash \bigcup(\mathrm{dom}(\!|\ \rho\ |\!)) = \mathrm{dom}(\bigcup \rho)$$
$$[X, Y]\ r, s : X \leftrightarrow Y \vdash \mathrm{dom}\, r \cup \mathrm{dom}\, s = \mathrm{dom}(r \cup s)$$
$$[X, Y]\ \rho : \mathbb{P}(X \leftrightarrow Y) \vdash \bigcup(\mathrm{ran}(\!|\ \rho\ |\!)) = \mathrm{ran}(\bigcup \rho)$$
$$[X, Y]\ r, s : X \leftrightarrow Y \vdash \mathrm{ran}\, r \cup \mathrm{ran}\, s = \mathrm{ran}(r \cup s)$$

Law 17.4 As a corollary of law 26.37 specialised to subsets, along with law 17.3, *dom* is subset order-preserving.

$$[X, Y]\ r, s : X \leftrightarrow Y \mid r \subseteq s \vdash \mathrm{dom}\, r \subseteq \mathrm{dom}\, s$$
$$[X, Y]\ r, s : X \leftrightarrow Y \mid r \subseteq s \vdash \mathrm{ran}\, r \subseteq \mathrm{ran}\, s$$

Law 17.5 Domain does not in general distribute through intersection. It distributes only when the relations are compatible (§20). Range does not in general distribute through intersection. It distributes only when the relations agree on their common ranges.

$$[X, Y]\ r, s : X \leftrightarrow Y \vdash \mathrm{dom}(r \cap s) \subseteq \mathrm{dom}\, r \cap \mathrm{dom}\, s$$
$$[X, Y]\ r, s : X \leftrightarrow Y \mid r \approx s \vdash \mathrm{dom}(r \cap s) = \mathrm{dom}\, r \cap \mathrm{dom}\, s$$
$$[X, Y]\ r, s : X \leftrightarrow Y \vdash \mathrm{ran}(r \cap s) \subseteq \mathrm{ran}\, r \cap \mathrm{ran}\, s$$
$$[X, Y]\ r, s : X \leftrightarrow Y \mid s \rhd \mathrm{ran}\, r = r \rhd \mathrm{ran}\, s \vdash \mathrm{ran}(r \cap s) = \mathrm{ran}\, r \cap \mathrm{ran}\, s$$

■ **Law 17.6** Domain does not in general distribute through set difference. It distributes only when the domain of the difference does not overlap the domain of the second relation.

$$[X, Y]\ r, s : X \leftrightarrow Y \vdash \mathrm{dom}\, r \setminus \mathrm{dom}\, s = \mathrm{dom}(r \setminus s) \setminus \mathrm{dom}\, s$$
$$[X, Y]\ r, s : X \leftrightarrow Y \mid \mathrm{disjoint}\langle \mathrm{dom}(r \setminus s), \mathrm{dom}\, s\rangle \vdash \mathrm{dom}\, r \setminus \mathrm{dom}\, s = \mathrm{dom}(r \setminus s)$$

Law 17.7 The domain and range of a relation are empty precisely when the relation is empty.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r = \varnothing \Leftrightarrow \mathrm{dom}\, r = \varnothing$$
$$[X, Y]\ r : X \leftrightarrow Y \vdash r = \varnothing \Leftrightarrow \mathrm{ran}\, r = \varnothing$$

Law 17.8 The domain of the full relation is the entire source set.

$$[X, Y]\ \vdash \mathrm{dom}(X \times Y) = X$$
$$[X, Y]\ \vdash \mathrm{ran}(X \times Y) = Y$$

———————————————————•

Chapter 18

Relation restriction

This chapter introduces some notaton for subsetting (binary) relations.

- domain restriction and subtraction: $(\_ \lhd \_)$, $(\_ \lhd\!\!\!\!- \_)$
- range restriction and subtraction: $(\_ \rhd \_)$, $(\_ \rhd\!\!\!\!- \_)$

Domain restriction

Intent

We can build a subset of a relation by considering it on only some subset of its domain.

Definition

domain restriction:

$$\begin{array}{l} \text{function } 65 \text{ rightassoc } (\_ \lhd \_) \\ \_ \lhd \_ \,[X, Y] == \lambda\, a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \bullet \{\ p : r \mid p.1 \in a\ \} \end{array}$$

Domain restriction, $a \lhd r$, gives a relation the same as r on a, and empty elsewhere. a need not be a subset of $\operatorname{dom} r$.

domain subtraction:

$$\begin{array}{l} \text{function } 65 \text{ rightassoc } (\_ \lhd\!\!\!\!- \_) \\ \_ \lhd\!\!\!\!- \_ \,[X, Y] == \lambda\, a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \bullet \{\ p : r \mid p.1 \notin a\ \} \end{array}$$

Domain subtraction, $a \lhd\!\!\!\!- r$, gives a relation that is empty on a, and the same as r elsewhere.

Examples

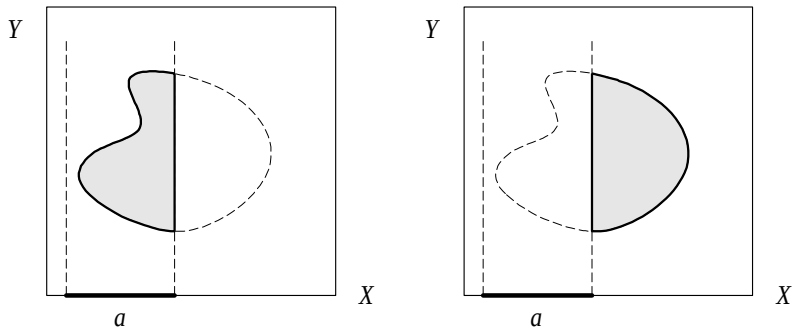


Figure 18.1 An example of domain restriction and subtraction.

1. Figure 18.1 shows an example of restriction.
2. $r = \{x_1 \mapsto y_1, x_1 \mapsto y_2, x_3 \mapsto y_2, x_5 \mapsto y_6\} \land \#\{x_1, x_3, x_5\} = 3$
   $\{x_1, x_5\} \lhd r = \{x_1 \mapsto y_1, x_1 \mapsto y_2, x_5 \mapsto y_6\}$
   $\{x_1, x_5\} \lhd\!\!\!\!- \, r = \{x_3 \mapsto y_2\}$
3. We can restrict the addition operator to just non-zero natural numbers
   $\mathbb{N}_1^{2\times} \lhd (\_ + \_)$
4. We can extract an interesting subrelation, involving only children's seating, where $seating : PERSON \leftrightarrow SEAT$, and $children : \mathbb{P}\, PERSON$
   $children \lhd seating$
5. We can restrict a sequence s to its first n elements
   $(1 \,..\, n) \lhd s$
6. We can extract a subsequence from a sequence s
   $\mathrm{squash}(a \lhd s)$
7. We can update a relation by removing a single element from its domain
   $r' = \{x\} \lhd\!\!\!\!- \, r$

Laws

Law 18.1 Restriction and subtraction are total surjections (§21).

$$[X, Y] \vdash (\_ \lhd \_)[X, Y] \in \mathbb{P}\, X \times (X \leftrightarrow Y) \twoheadrightarrow (X \leftrightarrow Y)$$
$$[X, Y] \vdash (\_ \lhd\!\!\!\!- \, \_)[X, Y] \in \mathbb{P}\, X \times (X \leftrightarrow Y) \twoheadrightarrow (X \leftrightarrow Y)$$

Law 18.2 Subtraction is the complement of restriction.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash a \lhd\!\!\!\!- r = (X \setminus a) \lhd r$$

Law 18.3 Restriction and subtraction together partition a relation.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \langle a \lhd r, a \lhd\!\!\!\!- r \rangle \text{ partition } r$$

Law 18.4 Domain restriction can be written as set intersection; domain subtraction can be written as set difference.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \mid \operatorname{ran} r \subseteq b \vdash a \lhd r = r \cap (a \times b)$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \mid \operatorname{ran} r \subseteq b \vdash a \lhd\!\!\!\!- r = r \setminus (a \times b)$$

■ **Law 18.5** Restriction distributes through generalised union on both its arguments.

$$[X, Y]\ \alpha : \mathbb{P}\,\mathbb{P}\, X;\ \rho : \mathbb{P}(X \leftrightarrow Y) \vdash \bigcup\{\ a : \alpha;\ r : \rho \bullet a \lhd r\ \} = \bigcup \alpha \lhd \bigcup \rho$$
$$[X, Y]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash (a \lhd r) \cup (b \lhd r) = (a \cup b) \lhd r$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \vdash (a \lhd r) \cup (a \lhd s) = a \lhd (r \cup s)$$

■ **Law 18.6** Restriction distributes through non-empty intersection on both its arguments.

$$[X, Y]\ \alpha : \mathbb{P}_1\,\mathbb{P}\, X;\ \rho : \mathbb{P}_1(X \leftrightarrow Y) \vdash \bigcap \alpha \lhd \bigcap \rho = \bigcap\{\ a : \alpha;\ r : \rho \bullet a \lhd r\ \}$$
$$[X, Y]\ a, b : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \vdash (a \cap b) \lhd (r \cap s) = (a \lhd r) \cap (b \lhd s)$$

Law 18.7 As a corollary of law 26.37 specialised to subsets, along with law 18.6, restriction is subset order-preserving on both its arguments.

$$[X, Y]\ a, b : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \mid a \subseteq b \land r \subseteq s \vdash a \lhd r \subseteq b \lhd s$$

■ **Law 18.8** Subtraction pseudo-distributes through union on its set argument, and distributes through union on its relation argument.

$$[X, Y]\ \alpha : \mathbb{P}\,\mathbb{P}\, X;\ \rho : \mathbb{P}(X \leftrightarrow Y) \vdash \bigcup\{\ a : \alpha;\ r : \rho \bullet a \lhd\!\!\!\!- r\ \} = \bigcap \alpha \lhd\!\!\!\!- \bigcup \rho$$
$$[X, Y]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash (a \lhd\!\!\!\!- r) \cup (b \lhd\!\!\!\!- r) = (a \cap b) \lhd\!\!\!\!- r$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \vdash (a \lhd\!\!\!\!- r) \cup (a \lhd\!\!\!\!- s) = a \lhd\!\!\!\!- (r \cup s)$$

■ **Law 18.9** Subtraction pseudo-distributes through non-empty intersection on its set argument, and distributes through non-empty intersection on its relation argument.

$$[X, Y]\ \alpha : \mathbb{P}_1 \mathbb{P} X;\ \rho : \mathbb{P}_1 (X \leftrightarrow Y) \vdash \bigcup \alpha \lhd \bigcap \rho = \bigcap \{\ a : \alpha;\ r : \rho \bullet a \lhd r\ \}$$
$$[X, Y]\ a, b : \mathbb{P} X;\ r : X \leftrightarrow Y \vdash (a \cup b) \lhd r = (a \lhd r) \cap (b \lhd r)$$
$$[X, Y]\ a : \mathbb{P} X;\ r, s : X \leftrightarrow Y \vdash a \lhd (r \cap s) = (a \lhd r) \cap (a \lhd s)$$

Law 18.10 As a corollary of law 26.40 specialised to subsets, along with law 18.9, restriction is subset order-reversing on its set argument. As a corollary of law 26.37 specialised to subsets, along with law 18.9, restriction is subset order-preserving on its relation argument.

$$[X, Y]\ a, b : \mathbb{P} X;\ r, s : X \leftrightarrow Y \mid a \subseteq b \land r \subseteq s \vdash b \lhd r \subseteq a \lhd s$$

■ **Law 18.11** If a binary function distributes through restriction on its relation argument, then it also distributes through subtraction.

$$[X, Y]\ f : (X \leftrightarrow Y)^{2\times} \to (X \leftrightarrow Y);\ a : \mathbb{P} X;\ r, s : X \leftrightarrow Y \mid$$
$$\forall a' : \mathbb{P} X;\ r', s' : X \leftrightarrow Y \bullet a' \lhd f(r', s') = f(a' \lhd r', a' \lhd s') \vdash$$
$$a \lhd f(r, s) = f(a \lhd r, a \lhd s)$$

■ **Law 18.12** A restricted relation is a subset of the relation. Equality with the empty set holds precisely when the domain restricting set is disjoint from the domain. Equality with r holds precisely when the domain restricting set is bigger than the domain.

$$[X, Y]\ a : \mathbb{P} X;\ r : X \leftrightarrow Y \vdash a \lhd r \subseteq r$$
$$[X, Y]\ a : \mathbb{P} X;\ r : X \leftrightarrow Y \vdash a \lhd r = \varnothing \Leftrightarrow a \cap \operatorname{dom} r = \varnothing$$
$$[X, Y]\ a : \mathbb{P} X;\ r : X \leftrightarrow Y \vdash a \lhd r = r \Leftrightarrow \operatorname{dom} r \subseteq a$$

Law 18.13 A subtracted relation is a subset of the relation. Equality with the empty set holds precisely when the domain subtracting set is bigger than the domain. Equality with r holds precisely when the domain subtractset is disjoint from the domain.

$$[X, Y]\ a : \mathbb{P} X;\ r : X \leftrightarrow Y \vdash a \lhd r \subseteq r$$
$$[X, Y]\ a : \mathbb{P} X;\ r : X \leftrightarrow Y \vdash a \lhd r = \varnothing \Leftrightarrow \operatorname{dom} r \subseteq a$$
$$[X, Y]\ a : \mathbb{P} X;\ r : X \leftrightarrow Y \vdash a \lhd r = r \Leftrightarrow a \cap \operatorname{dom} r = \varnothing$$

■ **Law 18.14** The part of the restricting or subtracting set outside the domain of the relation has no effect.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash a \lhd r = (a \cap \operatorname{dom} r) \lhd r$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash a \lhd\!\!\!- r = (a \cap \operatorname{dom} r) \lhd\!\!\!- r$$

■ **Law 18.15** Restricting twice is equivalent to restricting to the intersection; subtracting twice is equivalent to subtracting the union; restricting and subtracting is equivalent to restricting to the difference.

$$[X, Y]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash a \lhd b \lhd r = (a \cap b) \lhd r$$
$$[X, Y]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash a \lhd\!\!\!- b \lhd\!\!\!- r = (a \cup b) \lhd\!\!\!- r$$
$$[X, Y]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash a \lhd b \lhd\!\!\!- r = b \lhd\!\!\!- a \lhd r = (a \setminus b) \lhd r$$

Law 18.16 The domain of a restricted relation is the intersection of the original domain and the restriction set; the domain of subtracted relation is the difference of the original domain and the restriction set.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \operatorname{dom}(a \lhd r) = a \cap \operatorname{dom} r$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \operatorname{dom}(a \lhd\!\!\!- r) = \operatorname{dom} r \setminus a$$

Law 18.17 The range of a restricted relation is the image of the restriction set; it is also the range of the original relation less the upper shadow of the restriction set. The range of a subtracted relation is the image of all but the restriction set.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \operatorname{ran}(a \lhd r) = r(\!|\ a\ |\!)$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \operatorname{ran}(a \lhd\!\!\!- r) = \operatorname{ran} r \setminus \operatorname{upperShadow}\ r\ a$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \operatorname{ran}(a \lhd\!\!\!- r) = r(\!|\ \operatorname{dom} r \setminus a\ |\!)$$

———————————————•

Range restriction

Intent

We can build a subset of a relation by considering it on only some subset of its range.

Definition

range restriction:

$$\begin{array}{l}\text{function } 60 \text{ leftassoc } (\_ \rhd \_) \\ \_ \rhd \_ \, [X, Y] == \lambda\, r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \bullet \{\ p : r \mid p.2 \in b\ \}\end{array}$$

Range restriction, $r \rhd b$, gives a relation the same as r on b, and empty elsewhere. b need not be a subset of $\operatorname{ran} r$.

range subtraction:

$$\begin{array}{l}\text{function } 60 \text{ leftassoc } (\_ \rhd \_) \\ \_ \rhd \_ \, [X, Y] == \lambda\, r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \bullet \{\ p : r \mid p.2 \notin b\ \}\end{array}$$

Range subtraction, $r \rhd b$, gives a relation that is empty on b, and the same as r elsewhere.

Examples

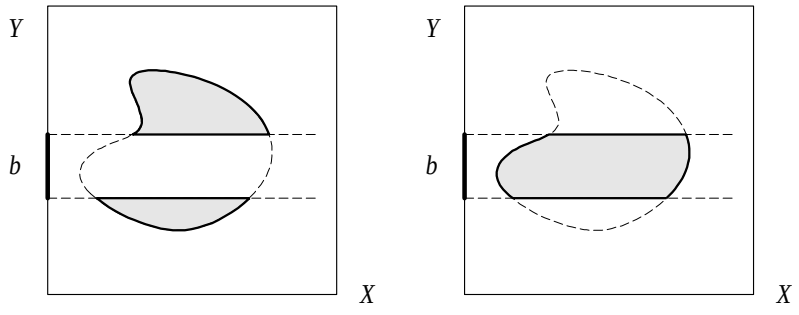


Figure 18.2 An example of range subtraction and restriction.

1. $r = \{x_1 \mapsto y_1, x_1 \mapsto y_2, x_3 \mapsto y_2, x_5 \mapsto y_6\} \land \#\{y_1, y_2, y_6\} = 3$
   $r \rhd \{y_2\} = \{x_1 \mapsto y_2, x_3 \mapsto y_2\}$
   $r \rhd\!\!> \{y_2\} = \{x_1 \mapsto y_1, x_5 \mapsto y_6\}$
2. We can filter a set of items from a sequence s
   $\mathrm{squash}(s \rhd b)$
3. We can extract an interesting subrelation, involving only non-first class seating, where $seating : PERSON \leftrightarrow SEAT$, and $firstClass : \mathbb{P}\, SEAT$
   $seating \rhd\!\!> firstClass$

Laws

Law 18.18 Range restriction laws can be derived as 'duals' of domain restriction laws, by using

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash r^{\sim} \rhd a = (a \lhd r)^{\sim}$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash r^{\sim} \rhd\!\!> a = (a \lhd\!\!< r)^{\sim}$$

The dual laws are not stated explicitly here.

Law 18.19 Brackets are not necessary when restricting both domain and range. (They have different precedences only because they have different associativities.)

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash (a \lhd r) \rhd b = a \lhd (r \rhd b) = a \lhd r \rhd b$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash (a \lhd r) \rhd\!\!> b = a \lhd (r \rhd\!\!> b) = a \lhd r \rhd\!\!> b$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash (a \lhd\!\!< r) \rhd b = a \lhd\!\!< (r \rhd b) = a \lhd\!\!< r \rhd b$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash (a \lhd\!\!< r) \rhd\!\!> b = a \lhd\!\!< (r \rhd\!\!> b) = a \lhd\!\!< r \rhd\!\!> b$$

Law 18.20 Domain and range restriction together act as set intersection.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash a \lhd r \rhd b = r \cap (a \times b)$$

---•

Chapter 19

Images, bounds, shadows

- upper and lower image
- upper and lower bound
- upper and lower shadow
- upper and lower singleton image

Upper and lower image

Intent

Image can be thought of as 'relational application'; applying a relation to a set, to yield the image set. Compare this with functional application, which applies a function to an element, to yield the image element (law 21.2).

Definition

upper image:

$$\mathrm{upperImage}[X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \lambda\, a : \mathbb{P}\, X \bullet \{\ y : Y \mid \exists\, x : a \bullet x \mapsto y \in r\ \}$$

The upper image of a set a through a relation r is the set of those range elements of r that are related to some element in a by r.

lower image:

$$\mathrm{lowerImage}[X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \lambda\, b : \mathbb{P}\, Y \bullet \{\ x : X \mid \exists\, y : b \bullet x \mapsto y \in r\ \}$$

The lower image of a set b through a relation r is the set of those domain elements of r that are related to some element in b by r.

Upper image is more familiarly written in Z as relational image, $r(\!|\ a\ |\!)$. In *ZRM* this is an undefinable kind of symbol that must be hardwired into Z's syntax; in *Standard* Z it is definable with an operator template paragraph.

relational image:

function $(\_(\!|\ \_\ |\!))$
$\_(\!|\ \_\ |\!)\ [X, Y] == \lambda\, r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \bullet \text{upperImage}\, r\ a$

Examples

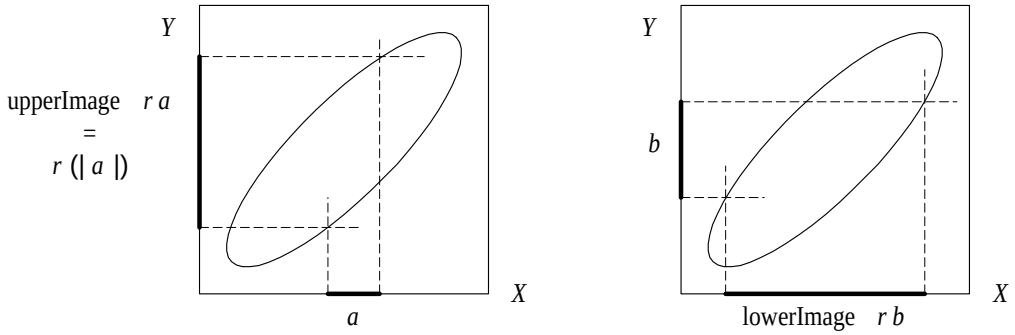


Figure 19.1 An example of upper image (relational image) and lower image.

1. $r = \{x_1 \mapsto y_1, x_1 \mapsto y_2, x_3 \mapsto y_2, x_5 \mapsto y_6\}$
   $r(\!|\ \{x_1, x_3\}\ |\!) = \{y_1, y_2\}$
2. Consider a register that relates houses to those people who own the house:
   owns : $HOUSE \leftrightarrow PERSON$
   We can extract everyone who owns the house *chezNous*:
   upperImage *owns* $\{chezNous\}$; or equivalently as *owns*$(\!|\ \{chezNous\}\ |\!)$
   We can extract every house owned by the person *fred*:
   lowerImage *owns* $\{fred\}$

Laws

Law 19.1 Lower image laws can be derived as 'duals' of upper image laws, and vice versa, by using

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \mathrm{lowerImage}(r^\sim)a = \mathrm{upperImage}\, r\ a = r(\!|\ a\ |\!)$$
$$[X, Y]\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash \mathrm{lowerImage}\, r\ b = \mathrm{upperImage}(r^\sim)b = (r^\sim)(\!|\ b\ |\!)$$

Law 19.2 Image is a total function. The uncurried form $\_(\!|\ \_\ |\!)$ is a total surjection (§21).

$$[X, Y]\ \vdash \mathrm{upperImage}[X, Y] \in (X \leftrightarrow Y) \to \mathbb{P}\, X \to \mathbb{P}\, Y$$
$$[X, Y]\ \vdash (\_(\!|\ \_\ |\!))[X, Y] \in (X \leftrightarrow Y) \times \mathbb{P}\, X \twoheadrightarrow \mathbb{P}\, Y$$

Law 19.3 The upper image of a function reduces to

$$[X, Y]\ a : \mathbb{P}\, X;\ f : X \nrightarrow Y \vdash f(\!|\ a\ |\!) = \{\ x : a \cap \mathrm{dom}\, f \bullet f\ x\ \}$$

■ **Law 19.4** Image distributes through union on its set and relation argument.

$$[X, Y]\ \rho : \mathbb{P}(X \leftrightarrow Y);\ \alpha : \mathbb{P}\,\mathbb{P}\, X \vdash \bigcup\{\ r : \rho;\ a : \alpha \bullet r(\!|\ a\ |\!)\ \} = \bigcup \rho(\!|\ \bigcup \alpha\ |\!)$$
$$[X, Y]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash r(\!|\ a\ |\!) \cup r(\!|\ b\ |\!) = r(\!|\ a \cup b\ |\!)$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \vdash r(\!|\ a\ |\!) \cup s(\!|\ a\ |\!) = (r \cup s)(\!|\ a\ |\!)$$

Law 19.5 As a corollary of law 26.37 specialised to subsets, along with law 19.4, image is subset order-preserving on its relation argument and on its set argument.

$$[X, Y]\ a, b : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \mid a \subseteq b \wedge r \subseteq s \vdash r(\!|\ a\ |\!) \subseteq s(\!|\ b\ |\!)$$

■ **Law 19.6** Lower image distributes through intersection on its set argument if the relation is a function (§21).

$$[X, Y]\ f : X \nrightarrow Y;\ a, b : \mathbb{P}\, Y \vdash$$
$$\quad \mathrm{lowerImage}\, f(a \cap b) = \mathrm{lowerImage}\, f\ \ a \cap \mathrm{lowerImage}\, f\ \ b$$

■ **Law 19.7** The part of a contained in the domain of relation r is contained in the lower image of its upper image through r.

The part of b contained in the range of relation r is contained in the upper image of its lower image through r. Equality holds when r is a function (§21).

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash a \cap \mathrm{dom}\, r \subseteq \mathrm{lowerImage}\, r(r(\!|\ a\ |\!))$$
$$[X, Y]\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash b \cap \mathrm{ran}\, r \subseteq r(\!|\ \mathrm{lowerImage}\, r\ \ b\ |\!)$$
$$[X, Y]\ f : X \nrightarrow Y;\ b : \mathbb{P}\, Y \vdash b \cap \mathrm{ran}\, f = f(\!|\ \mathrm{lowerImage}\, f\ \ b\ |\!)$$

■ **Law 19.8** Upper image of a domain restricted relation is the same as upper image through the restricted set.

$$[X, Y]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash (a \lhd r)(\!|\ b\ |\!) = r(\!|\ a \cap b\ |\!) = (b \lhd r)(\!|\ a\ |\!)$$
$$[X, Y]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash (a \lhd\!\!\!\!- r)(\!|\ b\ |\!) = r(\!|\ b \setminus a\ |\!)$$

■ **Law 19.9** Upper image of a range restricted relation is the same as the upper image, restricted.

$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y;\ r : X \leftrightarrow Y \vdash (r \rhd b)(\!|\ a\ |\!) = r(\!|\ a\ |\!) \cap b$$
$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y;\ r : X \leftrightarrow Y \vdash (r \rhd\!\!\!\!- b)(\!|\ a\ |\!) = r(\!|\ a\ |\!) \setminus b$$

■ **Law 19.10** Domain restriction by a set is a subset of range restriction by the upper image of that set; domain subtraction by a set is a superset of range subtraction by the upper image of that set.

$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash a \lhd r \subseteq r \rhd r(\!|\ a\ |\!)$$
$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash r \rhd\!\!\!\!- r(\!|\ a\ |\!) \subseteq a \lhd\!\!\!\!- r$$

Law 19.11 The upper image of r through the relation *first* is the domain of r; the upper image of r through the relation *second* is the range of r.

$$[X, Y]\ r : X \leftrightarrow Y \vdash \mathrm{first}(\!|\ r\ |\!) = \mathrm{dom}\, r$$
$$[X, Y]\ r : X \leftrightarrow Y \vdash \mathrm{second}(\!|\ r\ |\!) = \mathrm{ran}\, r$$

•

Upper and lower bound

Intent

The bound gives the range values which correspond to *all* domain values in the set a (contrast this with the image, which gives the range values which correspond to *some* domain value in the set a.)

Definition

upper bound:

$$\mathrm{upperBound}[X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \lambda\, a : \mathbb{P}\, X \bullet \{\ y : Y \mid \forall\, x : a \bullet x \mapsto y \in r\ \}$$

The upper bound of a set a through a relation r is the set of those elements of the target type of r that are related to all elements in a by r.

lower bound:

$$\mathrm{lowerBound}[X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \lambda\, b : \mathbb{P}\, Y \bullet \{\ x : X \mid \forall\, y : b \bullet x \mapsto y \in r\ \}$$

The lower bound of a set b through a relation r is the set of those elements of the source type of r that are related to all elements in b by r.

Examples

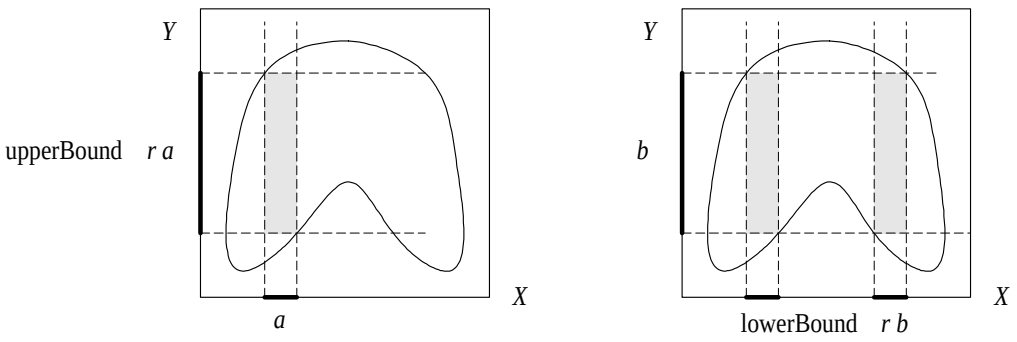


Figure 19.2 An example of upper bounds and of lower bounds of a set with respect to a relation.

The nature of the upper or lower bound is particularly clear when the relation is a total order (§26). The upper bound is then the set of all elements of the range of the order that follow all the elements of the set, and the lower bound is the set of all elements of the domain of the order that precede all the elements of the set.

For example:

$$r = \{x, y : 1 \mathinner{..} 10 \mid x \leq y\}$$
$$\mathrm{upperBound}\, r\{3, 5, 7\} = \{7, 8, 9, 10\}$$
$$\mathrm{lowerBound}\, r\{3, 5, 7\} = \{1, 2, 3\}$$

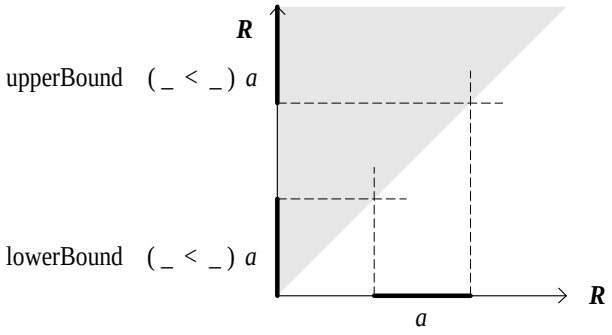


Figure 19.3 Example: lower bound and upper bound of a set a with respect to the order $(\_ < \_)$ over real numbers.

Similarly, if the relation is a partial order such as $\subset$ and we have some non-empty set of sets α, then the upper bound is the set of all sets of which all members of α are proper subsets, so

$$\alpha : \mathbb{P}_1 \mathbb{P}\, X \vdash \text{upperBound}(\_ \subset \_)\alpha = \{\ b : \mathbb{P}_1\, X \bullet b \cup \bigcup \alpha\ \}$$

and the lower bound is the set of all sets which are proper subsets of α, so

$$\alpha : \mathbb{P}_1 \mathbb{P}\, X \vdash \text{lowerBound}(\_ \subset \_)\alpha = \mathbb{P}(\bigcap \alpha) \setminus \{\bigcap \alpha\}$$

Further examples of image, bound and shadow are given following the definition of *shadow* below.

Laws

Law 19.12 Laws about lower bounds are 'duals' of laws about upper bound.

$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash \text{lowerBound}(r^{\sim})a = \text{upperBound}\, r\ a$$

Law 19.13 Bound is a total function.

$$[X, Y]\ \vdash \text{upperBound}[X, Y] \in (X \leftrightarrow Y) \to \mathbb{P}\, X \to \mathbb{P}\, Y$$

Law 19.14 An equivalent definition of upper bound, separated into empty and non-empty cases.

$$[X, Y]\ r : X \leftrightarrow Y \vdash \mathrm{upperBound}[X, Y] r\ \varnothing = Y$$
$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}_1\, X \vdash$$
$$\quad \mathrm{upperBound}\, r\ a = \{\ y : \mathrm{ran}\, r \mid \forall\, x : a \bullet x \mapsto y \in r\ \}$$

■ **Law 19.15** Bound distributes through intersection on its relation argument, and pseudo-distributes through union on its set argument:

$$[X, Y]\ \rho : \mathbb{P}(X \leftrightarrow Y);\ \alpha : \mathbb{P}\,\mathbb{P}\, X \vdash$$
$$\quad \mathrm{upperBound}(\bigcap \rho)(\bigcup \alpha) = \bigcap\{\ r : \rho;\ a : \alpha \bullet \mathrm{upperBound}\, r\ a\ \}$$
$$[X, Y]\ r, s : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash$$
$$\quad \mathrm{upperBound}(r \cap s) a = \mathrm{upperBound}\, r\ a \cap \mathrm{upperBound}\, s\ a$$
$$[X, Y]\ r : X \leftrightarrow Y;\ a, b : \mathbb{P}\, X \vdash$$
$$\quad \mathrm{upperBound}\, r(a \cup b) = \mathrm{upperBound}\, r\ a \cap \mathrm{upperBound}\, r\ b$$

Law 19.16 As a corollary of law 26.37, along with law 19.15, bound is subset order-preserving (§26) on its relation argument, and subset order-reversing (§26) on its set argument.

$$[X, Y]\ r, s : X \leftrightarrow Y;\ a, b : \mathbb{P}\, X \mid r \subseteq s \land a \subseteq b \vdash$$
$$\quad \mathrm{upperBound}\, r\ b \subseteq \mathrm{upperBound}\, s\ a$$

■ **Law 19.17** Upper bound of a range restriction is intersection of the upperbound.

$$[X, Y]\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y;\ a : \mathbb{P}_1\, X \vdash$$
$$\quad \mathrm{upperBound}(r \rhd b) a = (\mathrm{upperBound}\, r\ a) \cap b$$
$$[X, Y]\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y;\ a : \mathbb{P}_1\, X \vdash$$
$$\quad \mathrm{upperBound}(r \rhd b) a = (\mathrm{upperBound}\, r\ a) \setminus b$$

■ **Law 19.18** Upper bound of a domain restriction vanishes outside a subset of the restricting set.

$$[X, Y]\ r : X \leftrightarrow Y;\ b, a : \mathbb{P}\, X \vdash$$
$$\quad \mathrm{upperBound}(b \lhd r) a = (\ \text{if } a \subseteq b \text{ then } \mathrm{upperBound}\, r\ a \text{ else } \varnothing\ )$$
$$[X, Y]\ r : X \leftrightarrow Y;\ b, a : \mathbb{P}\, X \vdash$$
$$\quad \mathrm{upperBound}(b \lhd r) a = (\ \text{if } \mathrm{disjoint}\langle a, b\rangle \text{ then } \mathrm{upperBound}\, r\ a \text{ else } \varnothing\ )$$

Law 19.19 Being in the bound of a non-empty set is a stronger condition than being in the image. The two are equal only on the full relation.

$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}_1 X \vdash \mathrm{upperBound}\, r\ a \subseteq r(\!|\ a\ |\!)$$
$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}_1 X \vdash \mathrm{upperBound}\, r\ a = r(\!|\ a\ |\!) \Leftrightarrow a \lhd r = a \times Y$$

■ **Law 19.20** There is a pair of de Morgan-style laws relating image and bound through set complement.

$$[X, Y]\ r : X \leftrightarrow Y;\ a, c : \mathbb{P} X;\ b : \mathbb{P} Y \mid c \subseteq a \vdash$$
$$((a \times b) \setminus r)(\!|\ c\ |\!) = \mathrm{upperImage}((a \times b) \setminus r)c = b \setminus \mathrm{upperBound}\, r\ c$$
$$[X, Y]\ r : X \leftrightarrow Y;\ a, c : \mathbb{P}_1 X;\ b : \mathbb{P} Y \mid c \subseteq a \vdash$$
$$\mathrm{upperBound}((a \times b) \setminus r)c = b \setminus \mathrm{upperImage}\, r\ c = b \setminus (r(\!|\ c\ |\!))$$

By setting $a = X$ and $b = Y$ we obtain a form of this law using the generic parameter explicitly (figure 19.4), which can also be useful:

$$[X, Y]\ r : X \leftrightarrow Y;\ a : X \vdash$$
$$((X \times Y) \setminus r)(\!|\ a\ |\!) = \mathrm{upperImage}((X \times Y) \setminus r)a = Y \setminus \mathrm{upperBound}\, r\ a$$
$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}_1 X \vdash$$
$$\mathrm{upperBound}((X \times Y) \setminus r)a = Y \setminus \mathrm{upperImage}\, r\ a = Y \setminus (r(\!|\ a\ |\!))$$

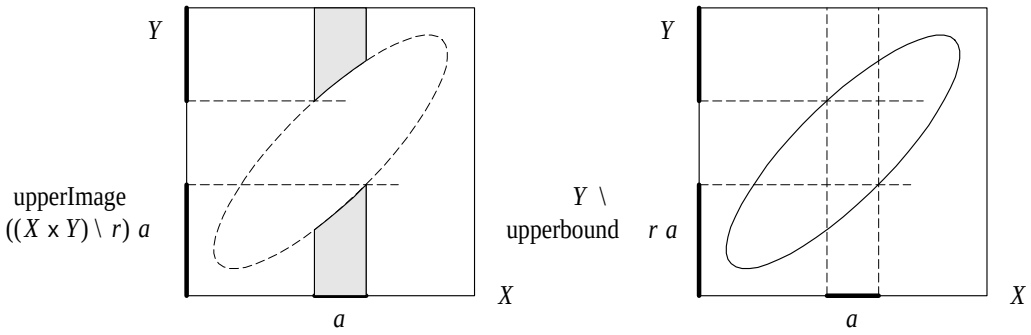


Figure 19.4 Illustration of one of the de Morgan-style laws relating *upperImage* and *upperBound* (law 19.20), in the form using the whole generic parameter sets.

Law 19.21 For an antisymmetric relation, at most *one* of the bounds is an element of the set.

$$[X]\ r : \mathrm{antisymmetric}\, X;\ a : \mathbb{P} X;\ x, y : X \mid \{x, y\} \subseteq a \cap \mathrm{upperBound}\, r\ a \vdash$$
$$x = y$$

Law 19.22 For an irreflexive relation, *none* of the bounds is an element of the set.

$$[X]\ r : \text{irreflexive}\, X;\ a : \mathbb{P}\, X \vdash \text{disjoint}\langle a, \text{upperBound}\, r\ a\rangle$$

Upper and lower shadow

Intent

The upper shadow gives the set of values of the target type of the relation whose corresponding domain values (if any) are wholly contained in the set.

Similarly the lower shadow gives the set of values of the source type of the relation whose corresponding range values (if any) are wholly contained in the set.

Definition

upper shadow:

$$\text{upperShadow}[X, Y] == \\ \lambda\, r : X \leftrightarrow Y \bullet \lambda\, a : \mathbb{P}\, X \bullet \{\ y : Y \mid \forall\, x : X \mid x \mapsto y \in r \bullet x \in a\ \}$$

The upper shadow of a set a through a relation r is the set of those elements y of the target type of r for which every x in the relation at that y is also in the set a.

lower shadow:

$$\text{lowerShadow}[X, Y] == \\ \lambda\, r : X \leftrightarrow Y \bullet \lambda\, b : \mathbb{P}\, Y \bullet \{\ x : X \mid \forall\, y : Y \mid x \mapsto y \in r \bullet y \in b\ \}$$

The lower shadow of a set b through a relation r is the set of those elements x of the source type of r for which every y in the relation at that x is also in the set b.

Examples

1. $\text{upperShadow}[1\,..\,10, 1\,..\,10](\_ < \_)\{1, 2, 5, 9, 10\} = \{1, 2, 3\}$
2. $\text{lowerShadow}[1\,..\,10, 1\,..\,10](\_ < \_)\{1, 2, 5, 9, 10\} = \{8, 9, 10\}$
3. Unlike the case with images and bounds, the extent of the generic parameter nearly always affects the value of this operation.

Further examples are given below.

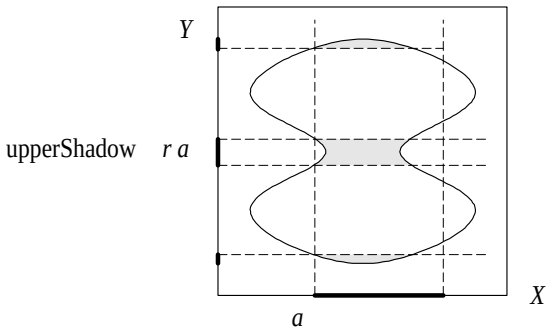


Figure 19.5 An example of upper shadow of a set with respect to a relation.

Laws

Law 19.23 Laws about lower shadows are 'duals' of laws about upper shadows.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \text{lowerShadow}(r^{\sim})a = \text{upperShadow}\, r\ a$$

Law 19.24 The upper shadow always contains the complement of the range of the relation.

$$\begin{array}{l}[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash \\ \quad \text{upperShadow}[X, Y]r\ a = \\ \qquad (Y \setminus \text{ran}\, r) \cup \{\ y : \text{ran}\, r \mid \forall\, x : X \mid x \mapsto y \in r \bullet x \in a\ \}\end{array}$$

Law 19.25 Shadow distributes through intersection on its set argument

$$\begin{array}{l}[X, Y]\ r : X \leftrightarrow Y;\ a, b : \mathbb{P}\ Y \vdash \\ \quad \text{upperShadow}[X, Y]r(a \cap b) = \\ \qquad \text{upperShadow}[X, Y]r\ a \cap \text{upperShadow}[X, Y]r\ b\end{array}$$

Law 19.26 Shadow pseudo-distributes through union on its relation argument

$$\begin{array}{l}[X, Y]\ r, s : X \leftrightarrow Y;\ b : \mathbb{P}\ Y \vdash \\ \quad \text{upperShadow}[X, Y](r \cup s)b = \\ \qquad \text{upperShadow}[X, Y]r\ b \cap \text{upperShadow}[X, Y]s\ b\end{array}$$

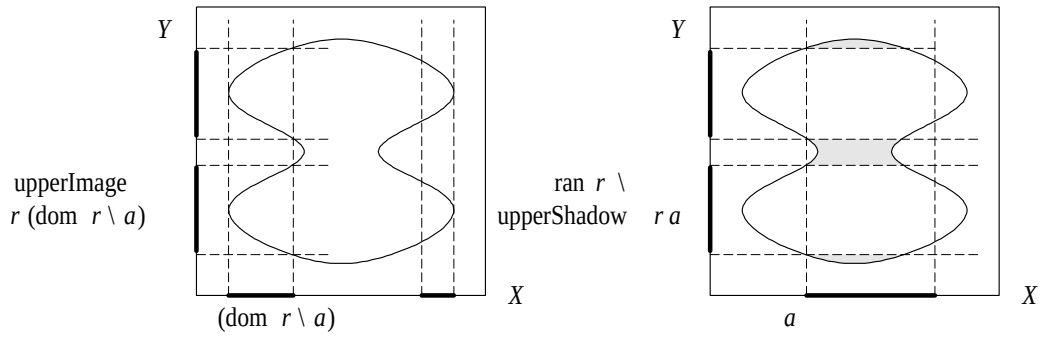


Figure 19.6 Illustration of one of the de Morgan-style laws relating *upperImage* and *upperShadow* (law 19.27).

Law 19.27 There is a pair of de Morgan-style laws relating image and shadow through set complement (figure 19.6).

$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash$$
$$r(\!|\ \mathrm{dom}\, r \setminus a\ |\!) = \mathrm{upperImage}\, r(\mathrm{dom}\, r \setminus a) = \mathrm{ran}\, r \setminus \mathrm{upperShadow}\, r\ a$$
$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash$$
$$\mathrm{ran}\, r \cap \mathrm{upperShadow}\, r(\mathrm{dom}\, r \setminus a) = \mathrm{ran}\, r \setminus \mathrm{upperImage}\, r\ a = \mathrm{ran}\, r \setminus (r(\!|\ a\ |\!))$$

There is an alternative form of this law, using the generic parameters explicitly, namely

$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash$$
$$\mathrm{upperShadow}\, r\ a = Y \setminus \mathrm{upperImage}\, r(X \setminus a)$$
$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash$$
$$\mathrm{upperImage}\, r\ a = Y \setminus \mathrm{upperShadow}\, r(X \setminus a)$$

Law 19.28 Laws about shadow inspired by refinement:

$$[X, Y]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash$$
$$Y \setminus b \subseteq \mathrm{ran}(a \lhd r) \Leftrightarrow \mathrm{upperShadow}\, r\ a \subseteq b$$
$$[X, Y]\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y;\ s : Y \leftrightarrow Y \vdash$$
$$r \fatsemi s \rhd \mathrm{ran}(a \lhd r) = r \fatsemi s \rhd \mathrm{upperShadow}\, r\ a$$

Larger examples

Consider the specification

$$[PERSON, EMPLOYER]$$

$$\begin{array}{|l} graduates : \mathbb{P}\, PERSON \\ ftse100 : \mathbb{P}\, EMPLOYER \\ hasWorkedFor : PERSON \leftrightarrow EMPLOYER \end{array}$$

then

- upperShadow $hasWorkedFor\ graduates$ is the set of employers who employ only graduates (including those who employ nobody)
- upperImage $hasWorkedFor\ graduates$ is the set of employers who have employed any graduates
- upperBound $hasWorkedFor\ graduates$ is the set of employers who have employed all the graduates
- lowerShadow $hasWorkedFor\ ftse100$ is the set of persons who work only for ftse100 firms (including those who have never worked at all)
- lowerImage $hasWorkedFor\ ftse100$ is the set of persons who have worked for any ftse100 firms
- lowerBound $hasWorkedFor\ ftse100$ is the set of persons who have worked for all the ftse100 firms

Consider the specification

$$[LEMMA]$$

$$\begin{array}{|l} myTheory : \mathbb{P}\, LEMMA \\ usedby : LEMMA \leftrightarrow LEMMA \end{array}$$

then

- upperShadow $usedby\ myTheory$ is the set of lemmas which use only those of myTheory (inluding those which use none at all)
- upperImage $usedby\ myTheory$ is the set of lemmas which use any of myTheory
- upperBound $usedby\ myTheory$ is the set of lemmas which use all those of myTheory

- lowerShadow *usedby myTheory* is the set of lemmas which are used only by myTheory (inluding those which are not used at all)
- lowerImage *usedby myTheory* is the set of lemmas which are used by any of myTheory
- lowerBound *usedby myTheory* is the set of lemmas which are used by all those of myTheory

•

Upper and lower singleton image

Intent

Provide a simplified version of image/bound in the case where the argument set is a singleton.

Definition

upper singleton image:

$$\text{successors}[X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \lambda\, x : X \bullet \{\ y : Y \mid x \mapsto y \in r\ \}$$

The upper singleton image of an element x through a relation r is the set of those range elements of r that are related to x by r. In graph terms (§24) it is the set of nodes at the end of the arcs leaving a vertex x.

lower singleton image:

$$\text{predecessors}[X, Y] == \lambda\, r : X \leftrightarrow Y \bullet \lambda\, y : Y \bullet \{\ x : X \mid x \mapsto y \in r\ \}$$

The lower image of an element y through a relation r is the set of those domain elements of r that are related to y by r. In graph terms (§24) it is the set of nodes at the start of the arcs entering a vertex y.

Laws

Law 19.29 Lower singleton image laws can be derived as 'duals' of upper singleton image laws, and vice versa, by using

$$[X, Y]\ x : X;\ r : X \leftrightarrow Y \vdash \text{successors}\ r\ x = \text{predecessors}\ r^{\sim}\ x$$

Law 19.30 The singleton images are related to the set images and bounds of singleton sets as follows:

$$[X, Y]\ x : X;\ r : X \leftrightarrow Y \vdash \\ \quad \text{successors}\, r\ x = \text{upperImage}\, r\{x\} = \text{upperBound}\, r\{x\}$$
$$[X, Y]\ r : X \leftrightarrow Y;\ y : Y \vdash \\ \quad \text{predecessors}\, r\ y = \text{lowerImage}\, r\{y\} = \text{lowerBound}\, r\{y\}$$

Law 19.31 The singleton images are related to the set images, bounds, and shadows of general sets as follows:

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \text{upperImage}\, r\ a = \bigcup\{\ x : a \bullet \text{successors}\, r\ x\ \}$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \text{upperBound}\, r\ a = \bigcap\{\ x : a \bullet \text{successors}\, r\ x\ \}$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y \vdash \text{upperShadow}\, r\ a = \{\ y : Y \mid \text{successors}\, r\ y \subseteq a\ \}$$

———————————————•

Chapter 20

Combining relations

There are some special purpose operations that are designed specifically for combining relations. Just as complicated sets may best be specified by building simple sets, and then combining them, so complicated relations may best be built from combinations of simpler ones.

- compatible relations: $(\_ \approx \_)$
- override: $(\_ \oplus \_)$, construct a relation that is equal to s on s's domain, and takes its values from a default relation r elsewhere, use $r \oplus s$.
- compose: $(\_ \fatsemi \_)$, $(\_ \circ \_)$, $(\_ \overrightarrow{\fatsemi} \_)$, $(\_ \overleftarrow{\fatsemi} \_)$, construct a relation from X to Z, by taking two 'steps', from X to an intermediate Y, then from Y to Z, use $r \fatsemi s$.
- merge: to construct a relation to pairs $Y \times Z$ from a relation to Y and another to Z, use $merge(r, s)$.
- split
- bicomposition

Compatible relations

Intent

Two relations are *compatible* if they agree on their common domain. Certain operations on relations (and particularly functions) require them to be compatible.

Definition

relation $(\_ \approx \_)$

$\_ \approx \_ [X, Y] == \{ r, s : X \leftrightarrow Y \mid \mathrm{dom}\, r \lhd s = \mathrm{dom}\, s \lhd r \}$

Examples

1. The empty relation is compatible with all relations
2. *dom* distributes through $\cap$ of compatible relations (law 17.5)
3. The union of two functions is a function precisely when they are compatible (law 21.8)

Laws

Law 20.1 Compatibility is reflexive ($r \approx r$) and symmetric ($r \approx s \Rightarrow s \approx r$).

$$[X, Y] \ \vdash (\_ \approx \_)[X, Y] \in \text{reflexive}(X \leftrightarrow Y) \cap \text{symmetric}(X \leftrightarrow Y)$$

Law 20.2 Relations with disjoint domains are compatible.

$$[X, Y] \ r, s : X \leftrightarrow Y \mid \text{disjoint}\langle \text{dom}\, r, \text{dom}\, s \rangle \vdash r \approx s$$

Law 20.3 The union of compatible relations is unchanged on their common domain. The intersection of compatible relations results in a restriction to their common domain.

$$[X, Y] \ r, s : X \leftrightarrow Y \mid r \approx s \vdash \\ \quad (\text{dom}\, r \cap \text{dom}\, s) \lhd (r \cup s) = (\text{dom}\, r \cap \text{dom}\, s) \lhd r$$
$$[X, Y] \ r, s : X \leftrightarrow Y \mid r \approx s \vdash r \cap s = \text{dom}\, s \lhd r = \text{dom}\, r \lhd s$$

Law 20.4 An overridden relation is compatible with the overriding relation.

$$[X, Y] \ r, s : X \leftrightarrow Y \vdash r \oplus s \approx s$$

•

Override

Intent

$r \oplus s$ has the same values as s on the domain of s, and the values of r elsewhere.

Definition

function 50 leftassoc $(\_ \oplus \_)$

$\_ \oplus \_ [X, Y] == \lambda\, r, s : X \leftrightarrow Y \bullet (\operatorname{dom} s \lhd r) \cup s$

Examples

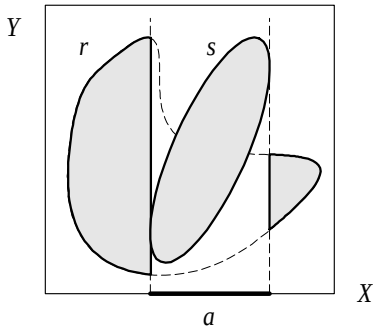


Figure 20.1 An example of relational overriding.

1. Figure 20.1 shows an example of relational overriding.
2. We can 'update' a relation by changing a single element
   $r' = r \oplus \{x \mapsto y\}$
3. We can make a relation total on its source set (so that $\operatorname{dom} r = X$) by overriding the full relation with r
   $r_t = (X \times Y) \oplus r$
4. We can make a relation total on its source set by overriding the relation that maps every x to some particular y, with r
   $r_t = (X \times \{y\}) \oplus r$
5. We can make a homogeneous relation total by overriding the identity relation with r
   $r_t = \operatorname{id} X \oplus r$

6. The *abs* function, that maps numbers to their absolute value, can be defined by overriding
   $\mathrm{id}\,\mathbb{R} \oplus (\ \lambda\, x : \mathbb{R}_{-} \bullet -x\ ) \subseteq \mathrm{abs}$

Laws

Law 20.5 Relational overriding is a total surjection (§21).

$$[X, Y] \ \vdash (\_ \oplus \_)[X, Y] \in (X \leftrightarrow Y)^{2\times} \twoheadrightarrow X \leftrightarrow Y$$

Law 20.6 Relational overriding is idempotent.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r \oplus r = r$$

Law 20.7 Relational overriding is closed, associative, and has the empty set as its identity element. Hence it forms a monoid (§23) over relations. It similarly forms a monoid over partial functions (§21).

$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash \\ \quad \exists\, \mathrm{Monoid}[X \leftrightarrow Y] \bullet g = a \leftrightarrow b \land (\_ \diamond \_) = (\_ \oplus \_) \land e = \varnothing$$
$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash \\ \quad \exists\, \mathrm{Monoid}[X \leftrightarrow Y] \bullet g = a \nrightarrow b \land (\_ \diamond \_) = (\_ \oplus \_) \land e = \varnothing$$

Law 20.8 Relational overriding reduces to union when the relations are compatible. Hence it forms an abelian monoid (§23) over compatible relations. It similarly forms an abelian monoid over compatible partial functions (§21).

$$[X, Y]\ r, s : X \leftrightarrow Y \vdash r \approx s \Leftrightarrow r \oplus s = r \cup s$$
$$[X, Y]\ \rho : \mathbb{P}(X \leftrightarrow Y) \mid (\forall\, r, s : \rho \bullet r \approx s) \vdash \\ \quad \exists\, \mathrm{AbelianMonoid}[X \leftrightarrow Y] \bullet g = \rho \land (\_ \diamond \_) = (\_ \oplus \_) \land e = \varnothing$$
$$[X, Y]\ \rho : \mathbb{P}(X \nrightarrow Y) \mid (\forall\, r, s : \rho \bullet r \approx s) \vdash \\ \quad \exists\, \mathrm{AbelianMonoid}[X \leftrightarrow Y] \bullet g = \rho \land (\_ \diamond \_) = (\_ \oplus \_) \land e = \varnothing$$

■ **Law 20.9** Relational overriding distributes through union and through non-empty intersections.

$$[X, Y]\ \rho : \mathbb{P}(X \leftrightarrow Y);\ s : X \leftrightarrow Y \vdash \bigcup\{\ r : \rho \bullet r \oplus s\ \} = \bigcup \rho \oplus s$$
$$[X, Y]\ r, r', s : X \leftrightarrow Y \vdash (r \oplus s) \cup (r' \oplus s) = (r \cup r') \oplus s$$
$$[X, Y]\ \rho : \mathbb{P}_1(X \leftrightarrow Y);\ s : X \leftrightarrow Y \vdash \bigcap \rho \oplus s = \bigcap\{\ r : \rho \bullet r \oplus s\ \}$$
$$[X, Y]\ r, r', s : X \leftrightarrow Y \vdash (r \cap r') \oplus s = (r \oplus s) \cap (r' \oplus s)$$

Law 20.10 Relational overriding is subset order-preserving (§26) on its first argument.

$$[X, Y]\ r, r', s : X \leftrightarrow Y \mid r \subseteq r' \vdash r \oplus s \subseteq r' \oplus s$$

■ **Law 20.11** The domain of an overriding is the union of the separate domains.

$$[X, Y]\ r, s : X \leftrightarrow Y \vdash \operatorname{dom}(r \oplus s) = \operatorname{dom} r \cup \operatorname{dom} s$$

■ **Law 20.12** Domain restriction and subtraction distribute through overriding.

$$[X, Y]\ a : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \vdash a \lhd (r \oplus s) = (a \lhd r) \oplus (a \lhd s)$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \vdash a \lhd\!\!\!- (r \oplus s) = (a \lhd\!\!\!- r) \oplus (a \lhd\!\!\!- s)$$

■ **Law 20.13** A restriction outside the domain of an overriding relation is independent of that relation. A restriction confined within the domain of an overriding relation depends on only that relation. A domain subtraction covering the domain of an overriding relation is independent of that relation.

$$[X, Y]\ a : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \mid \operatorname{disjoint}\langle a, \operatorname{dom} s\rangle \vdash a \lhd (r \oplus s) = a \lhd r$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \mid a \subseteq \operatorname{dom} s \vdash a \lhd (r \oplus s) = a \lhd s$$
$$[X, Y]\ a : \mathbb{P}\, X;\ r, s : X \leftrightarrow Y \mid \operatorname{dom} s \subseteq a \vdash a \lhd\!\!\!- (r \oplus s) = a \lhd\!\!\!- r$$

Law 20.14 Range restriction and subtraction do not in general distribute through overriding. Range restriction does distribute through overriding if the restricting set is larger than the range of the overriding relation. Range subtraction does distribute through overriding if the restricting set is disjoint from the range of the overriding relation.

$$[X, Y]\ r, s : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash (r \oplus s) \rhd b \subseteq (r \rhd b) \oplus (s \rhd b)$$
$$[X, Y]\ r, s : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash (r \oplus s) \rhd\!\!\!- b \subseteq (r \rhd\!\!\!- b) \oplus (s \rhd\!\!\!- b)$$
$$[X, Y]\ r, s : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \mid \operatorname{ran} s \subseteq b \vdash$$
$$(r \oplus s) \rhd b = (r \rhd b) \oplus (s \rhd b) = (r \rhd b) \oplus s$$
$$[X, Y]\ r, s : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \mid \operatorname{disjoint}\langle b, \operatorname{ran} s\rangle \vdash$$
$$(r \oplus s) \rhd\!\!\!- b = (r \rhd\!\!\!- b) \oplus (s \rhd\!\!\!- b) = (r \rhd\!\!\!- b) \oplus s$$

■ **Law 20.15** The upper image of an overriding is

$$[X, Y]\ r, s : X \leftrightarrow Y;\ a : \mathbb{P}\, X \vdash (r \oplus s)(\!|\ a\ |\!) = r(\!|\ a \setminus \operatorname{dom} s\ |\!) \cup s(\!|\ a\ |\!)$$

———————————————•

Composition

Intent

$r \fatsemi s$ captures the notion of 'apply r, then apply s to the result'.

Definition

relational composition:

$$\begin{aligned}&\text{function } 40 \text{ leftassoc } (\_ \fatsemi \_)\\ &\_ \fatsemi \_ \ [X, Y, Z] == \lambda\, r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \bullet\\ &\qquad\qquad \{\ x : X;\ z : Z \mid (\exists\, y : Y \bullet x \mapsto y \in r \land y \mapsto z \in s)\ \}\end{aligned}$$

functional composition:

$$\begin{aligned}&\text{function } 40 \text{ leftassoc } (\_ \circ \_)\\ &\_ \circ \_ \ [X, Y, Z] == \lambda\, r : Y \leftrightarrow Z;\ s : X \leftrightarrow Y \bullet s \fatsemi r\end{aligned}$$

Examples

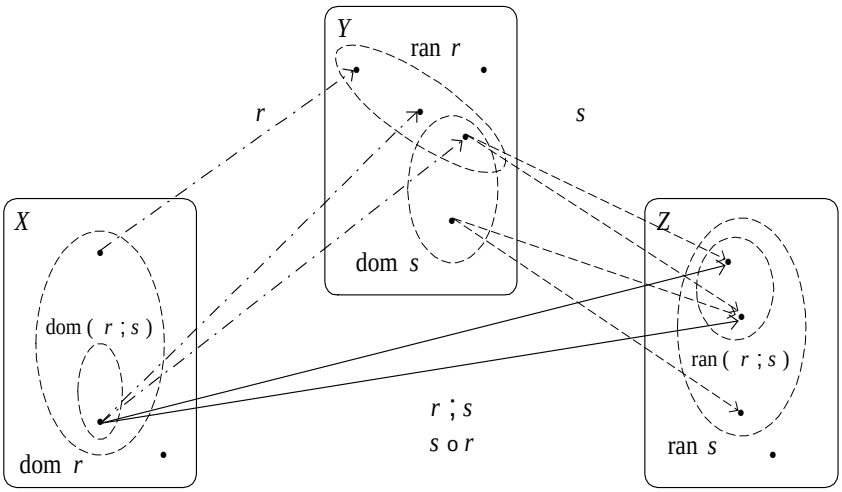


Figure 20.2 An example of relational composition.

1. Figure 20.2 shows an example of relational composition.

2. We define relational iteration in terms of relational composition
   $r^{n+1} = r \fatsemi r^n$
3. We can use functional composition to reduce the number of brackets needed to write repeated function applications
   $(f(g(h\ x))) = (f \circ g \circ h)x$

Laws

Law 20.16 Composition is a total surjection (§21).

$$[X, Y, Z] \vdash (\_ \fatsemi \_)[X, Y, Z] \in (X \leftrightarrow Y) \times (Y \leftrightarrow Z) \twoheadrightarrow (X \leftrightarrow Z)$$
$$[X, Y, Z] \vdash (\_ \circ \_)[X, Y, Z] \in (Y \leftrightarrow Z) \times (X \leftrightarrow Y) \twoheadrightarrow (X \leftrightarrow Z)$$

Law 20.17 Composition is closed, associative, and has the identity relation as its identity element, so forms a monoid (§23) over homogeneous relations. It similarly forms a monoid over homogeneous partial functions (§21), homogeneous total functions (§21), homogeneous surjections (§21), and homogeneous injections (§21).

$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \leftrightarrow a \wedge (\_ \diamond \_) = (\_ \fatsemi \_) \wedge e = \mathrm{id}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \pfun a \wedge (\_ \diamond \_) = (\_ \fatsemi \_) \wedge e = \mathrm{id}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \fun a \wedge (\_ \diamond \_) = (\_ \fatsemi \_) \wedge e = \mathrm{id}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \twoheadrightarrow a \wedge (\_ \diamond \_) = (\_ \fatsemi \_) \wedge e = \mathrm{id}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \rightarrowtail a \wedge (\_ \diamond \_) = (\_ \fatsemi \_) \wedge e = \mathrm{id}\, a$$

Law 20.18 Composition is closed, associative, has the identity relation as its identity element, and has the inverse relation as its inverse function, so forms a group (§23) over homogeneous bijections.

$$[X]\ a : \mathbb{P}\, X \vdash$$
$$\exists\, \mathrm{Group}[X \leftrightarrow X] \bullet g = a \rightarrowtail a \wedge (\_ \diamond \_) = (\_ \fatsemi \_) \wedge e = \mathrm{id}\, a \wedge inv = (\_^\sim)$$

Law 20.19 Composition with the full relation:

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ a : \mathbb{P}\, Y;\ b : \mathbb{P}\, Z \vdash r \fatsemi (a \times b) = \mathrm{dom}(r \rhd a) \times b$$
$$[X, Y, Z]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y;\ r : Y \leftrightarrow Z \vdash (a \times b) \fatsemi r = a \times \mathrm{ran}(b \lhd r)$$

Law 20.20 Composition with the empty relation yields the empty relation.

$$[X, Y, Z]\ r : X \leftrightarrow Y \vdash r \fatsemi \varnothing[Y \times Z] = \varnothing[X \times Z]$$
$$[X, Y, Z]\ r : Y \leftrightarrow Z \vdash \varnothing[X \times Y] \fatsemi r = \varnothing[X \times Z]$$

■ **Law 20.21** Composition distributes through generalised union.

$$[X, Y, Z]\ \rho : \mathbb{P}(X \leftrightarrow Y);\ \sigma : \mathbb{P}(Y \leftrightarrow Z) \vdash$$
$$\quad \bigcup \{\ r : \rho;\ s : \sigma \bullet r \fatsemi s\ \} = \bigcup \rho \fatsemi \bigcup \sigma$$
$$[X, Y, Z]\ r, r' : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash (r \fatsemi s) \cup (r' \fatsemi s) = (r \cup r') \fatsemi s$$
$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s, s' : Y \leftrightarrow Z \vdash (r \fatsemi s) \cup (r \fatsemi s') = r \fatsemi (s \cup s')$$

Law 20.22 As a corollary of law 26.37 specialised to subsets, along with law 20.21, composition is subset order-preserving on both arguments.

$$[X, Y, Z]\ r, r' : X \leftrightarrow Y;\ s, s' : Y \leftrightarrow Z \mid r \subseteq r' \land s \subseteq s' \vdash r \fatsemi s \subseteq r' \fatsemi s'$$

Law 20.23 Inverting a composition is the same as composing the inversions in the other order.

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash (r \fatsemi s)^{\sim} = s^{\sim} \fatsemi r^{\sim}$$

■ **Law 20.24** The composition of two relations has no reflexive pairs (§24) precisely when one relation and the other's inverse are disjoint.

$$[X, Y]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow X \vdash r \cap s^{\sim} = \varnothing \Leftrightarrow (r \fatsemi s) \cap \mathrm{id}\, X = \varnothing$$

■ **Law 20.25** The pairs of a composition are precisely those where the upper singleton image of the domain element and the lower singleton image of the range element share at least one element (the shared elements providing the necessary link).

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash$$
$$\quad r \fatsemi s = \{\ x : X;\ z : Z \mid \mathrm{successors}\, r\ x \cap \mathrm{predecessors}\, s\ z \neq \varnothing\ \}$$

■ **Law 20.26** The domain of a composition $r \fatsemi s$ is the lower image of the domain of s through r. The range of a composition $r \fatsemi s$ is the upper image of the range of r through s.

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash \mathrm{dom}(r \fatsemi s) = \mathrm{lowerImage}\, r(\mathrm{dom}\, s)$$
$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash \mathrm{ran}(r \fatsemi s) = s(\!|\, \mathrm{ran}\, r\ |\!)$$

Law 20.27 As a corollary of laws 18.17 and 20.26, the domain and range of a composition can be expressed in terms of restriction.

$$[X, Y, Z] \; r : X \leftrightarrow Y; \; s : Y \leftrightarrow Z \vdash \operatorname{dom}(r \fatsemi s) = \operatorname{dom}(r \rhd \operatorname{dom} s)$$
$$[X, Y, Z] \; r : X \leftrightarrow Y; \; s : Y \leftrightarrow Z \vdash \operatorname{ran}(r \fatsemi s) = \operatorname{ran}(\operatorname{ran} r \lhd s)$$

Law 20.28 As a further corollary of laws 18.17 and 20.26, composing a relation with its own inverse does not shrink its domain.

$$[X, Y] \; r : X \leftrightarrow Y \vdash \operatorname{dom} r = \operatorname{dom}(r \fatsemi r^\sim)$$
$$[X, Y] \; r : X \leftrightarrow Y \vdash \operatorname{ran} r = \operatorname{ran}(r^\sim \fatsemi r)$$

■ **Law 20.29** Restriction associates with composition.

$$[X, Y, Z] \; a : \mathbb{P}\, X; \; r : X \leftrightarrow Y; \; s : Y \leftrightarrow Z; \; b : \mathbb{P}\, Z \vdash$$
$$a \lhd (r \fatsemi s) \rhd b = (a \lhd r) \fatsemi (s \rhd b)$$
$$[X, Y, Z] \; a : \mathbb{P}\, X; \; r : X \leftrightarrow Y; \; s : Y \leftrightarrow Z; \; b : \mathbb{P}\, Z \vdash$$
$$a \lhd\!\!\cdot (r \fatsemi s) \rhd\!\!\cdot b = (a \lhd\!\!\cdot r) \fatsemi (s \rhd\!\!\cdot b)$$

■ **Law 20.30** Restricting the range of the first relation in a composition is the same as restricting the domain of the second; restriction pseudo-distributes through composition.

$$[X, Y, Z] \; r : X \leftrightarrow Y; \; b : \mathbb{P}\, Y; \; s : Y \leftrightarrow Z \vdash (r \rhd b) \fatsemi s = r \fatsemi (b \lhd s)$$
$$[X, Y, Z] \; r : X \leftrightarrow Y; \; b : \mathbb{P}\, Y; \; s : Y \leftrightarrow Z \vdash (r \rhd\!\!\cdot b) \fatsemi s = r \fatsemi (b \lhd\!\!\cdot s)$$

■ **Law 20.31** Nested images can be written as composition.

$$[X, Y, Z] \; a : \mathbb{P}\, X; \; r : X \leftrightarrow Y; \; s : Y \leftrightarrow Z \vdash (r \fatsemi s)(\!|\, a \,|\!) = s(\!|\, r(\!|\, a \,|\!) \,|\!)$$

■ **Law 20.32** Overriding then composing on the right is a subset of the overriding of the individual compositions.

$$[X, Y, Z] \; r, s : X \leftrightarrow Y; \; t : Y \leftrightarrow Z \vdash (r \oplus s) \fatsemi t \subseteq (r \fatsemi t) \oplus (s \fatsemi t)$$

Composition on the right does not in general distribute through overriding.

For example:

$$r = X \times b;\ s = X \times (Y \setminus b);\ t = b \times Z$$
$$(r \oplus s) \fatsemi t = s \fatsemi t = \varnothing$$
$$(r \fatsemi t) \oplus (s \fatsemi t) = (X \times Z) \oplus \varnothing = X \times Z$$

Law 20.33 Composition on the right distributes through overriding when the range of the overriding relation s is smaller than the domain of t.

$$[X, Y, Z]\ r, s : X \leftrightarrow Y;\ t : Y \leftrightarrow Z \mid \operatorname{ran} s \subseteq \operatorname{dom} t \vdash$$
$$(r \oplus s) \fatsemi t = (r \fatsemi t) \oplus (s \fatsemi t)$$

■ **Law 20.34** Overriding then composing on the left is a superset of the overriding of the individual compositions.

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s, t : Y \leftrightarrow Z \vdash (r \fatsemi s) \oplus (r \fatsemi t) \subseteq r \fatsemi (s \oplus t)$$

Composition on the left does not in general distribute through overriding.

For example:

$$r = X \times Y;\ s = b \times c;\ t = (Y \setminus b) \times (Z \setminus c)$$
$$(r \fatsemi s) \oplus (r \fatsemi t) = (X \times c) \oplus (X \times (Z \setminus c)) = X \times (Z \setminus c)$$
$$r \fatsemi (s \oplus t) = r \fatsemi ((b \times c) \cup ((Y \setminus b) \times (Z \setminus c)))$$
$$= (X \times c) \cup (X \times (Z \setminus c)) = X \times Z$$

Law 20.35 The name 'functional composition' is used for $\circ$ because this is the order of presentation of the arguments that appears natural when they are functions.

$$[X, Y, Z]\ f : Y \to Z;\ g : X \to Y;\ x : X \vdash (f \circ g)x = f(g\ x)$$

———————————————•

Demonic composition

Intent

Whereas composition gives a result if there is *some* path from the source to the target, demonic composition requires *all* paths to be present. This is more robust under refinement, since resolution of non-determinism will not leave an end-to-end path broken.

Definition

right demonic composition:

$$\begin{array}{l}\text{function } 40 \text{ leftassoc } (\_ \overset{\rightarrow}{\circ}_9 \_) \\ \_ \overset{\rightarrow}{\circ}_9 \_ [X, Y, Z] == \lambda\, r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \bullet \\ \qquad\qquad \{\ x : X;\ z : Z \mid (\forall\, y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in s)\ \}\end{array}$$

left demonic composition:

$$\begin{array}{l}\text{function } 40 \text{ leftassoc } (\_ \overset{\leftarrow}{\circ}_9 \_) \\ \_ \overset{\leftarrow}{\circ}_9 \_ [X, Y, Z] == \lambda\, r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \bullet \\ \qquad\qquad \{\ x : X;\ z : Z \mid (\forall\, y : Y \mid y \mapsto z \in s \bullet x \mapsto y \in r)\ \}\end{array}$$

Examples

Consider $r = \{a \mapsto c, a \mapsto d, b \mapsto d\}$, $s = \{c \mapsto e, c \mapsto f, d \mapsto f\}$. Then $r \fatsemi s = \{a \mapsto e, a \mapsto f, b \mapsto f\}$. However, $r \overset{\rightarrow}{\circ}_9 s = \{a \mapsto f, b \mapsto f\}$; the maplet $a \mapsto e$ is not present here because there is no path via d (a refinement that resolved non-determinism by dropping $a \mapsto c$ would find that e is no longer reachable from a). Also $r \overset{\leftarrow}{\circ}_9 s = \{a \mapsto e, a \mapsto f\}$; the maplet $b \mapsto f$ is not present because there is no path to f via c.

Note that demonic composition is not associative.

Laws

Law 20.36 Laws about left demonic composition are duals of laws about right demonic composition.

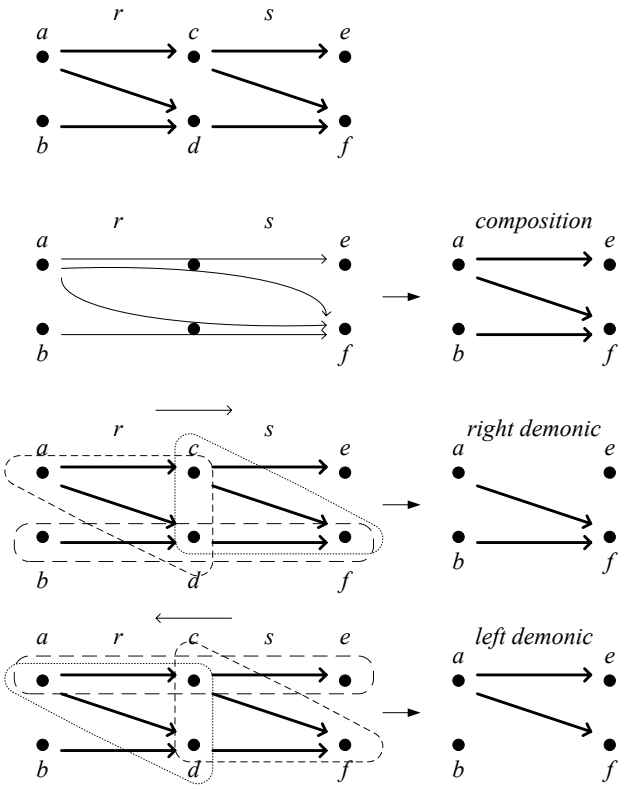


Figure 20.3 An example of right and left demonic composition. In the *right* case, a can reach both c and d, which can both reach f (but not e), so we get $a \mapsto f$ (but not $a \mapsto e$). In the *left* case, f can be reached from both c and d, which can both be reached from reach a (but not b), so we get $a \mapsto f$ (but not $b \mapsto f$).

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash (r \overleftarrow{\fatsemi} s)^\sim = s^\sim \overrightarrow{\fatsemi} r^\sim$$

■ **Law 20.37** Right demonic composition pseudo-distributes through union on its first argument and distributes through intersection on its second argument

$$[X, Y, Z]\ r, r' : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash (r \cup r') \overrightarrow{\fatsemi} s = (r \overrightarrow{\fatsemi} s) \cap (r' \overrightarrow{\fatsemi} s)$$
$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s, s' : Y \leftrightarrow Z \vdash r \overrightarrow{\fatsemi} (s \cap s') = (r \overrightarrow{\fatsemi} s) \cap (r \overrightarrow{\fatsemi} s')$$

Law 20.38 As a corollary of law 26.40 specialised to subsets, along with law 20.36, right demonic composition is subset order-reversing on its first argument. As a corollary of law 26.37 specialised to subsets, along with law 20.36, right demonic composition is subset order-preserving on its second argument.

$$[X, Y, Z]\ r, r' : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \mid r \subseteq r' \vdash r' \overset{\rightarrow}{\fatsemi} s \subseteq r \overset{\rightarrow}{\fatsemi} s$$
$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s, s' : Y \leftrightarrow Z \mid s \subseteq s' \vdash r \overset{\rightarrow}{\fatsemi} s \subseteq r \overset{\rightarrow}{\fatsemi} s'$$

■ **Law 20.39** The pairs of a right demonic composition are precisely those where the upper singleton image of the domain element falls completely within the lower singleton image of the range element.

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash$$
$$r \overset{\rightarrow}{\fatsemi} s = \{\ x : X;\ z : Z \mid \mathrm{successors}\, r\ x \subseteq \mathrm{predecessors}\, s\ z\ \}$$

■ **Law 20.40** The domain of a right demonic composition always includes the complement of the domain of its first argument.

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash$$
$$r \overset{\rightarrow}{\fatsemi} s = ((X \setminus \mathrm{dom}\, r) \times Z) \cup \mathrm{dom}\, r \lhd (r \overset{\rightarrow}{\fatsemi} s)$$

■ **Law 20.41** If the first argument is a function, right demonic composition on that domain reduces to composition.

$$[X, Y, Z]\ f : X \pfun Y;\ s : Y \leftrightarrow Z \vdash$$
$$f \overset{\rightarrow}{\fatsemi} s = ((X \setminus \mathrm{dom}\, f) \times Z) \cup (f \fatsemi s)$$

■ **Law 20.42** If the second argument is an inverse function, right demonic composition is made with the functional part of the first argument.

$$[X, Y, Z]\ r : X \pfun Y;\ s : Y \leftrightarrow Z \mid s^{\sim} \in Z \pfun Y \vdash$$
$$r \overset{\rightarrow}{\fatsemi} s = ((X \setminus \mathrm{dom}\, r) \times Z) \cup \{\ x : X \mid r\ \mathrm{functionalAt}\ x\ \} \lhd (r \fatsemi s)$$

■ **Law 20.43** There is a de Morgan style law relating right demonic composition and composition.

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash r \overset{\rightarrow}{\fatsemi} s = (X \times Z) \setminus (r \fatsemi ((Y \times Z) \setminus s))$$

———————————————•

Merge and split

Intent

merge takes two relations, and merges them into a single relation to pairs. *split* takes a single relation to pairs, and splits it into a pair of relations.

Definition

merge:

$$\begin{array}{l}\text{merge}[X, Y, Z] == \\ \quad \lambda\, r : X \leftrightarrow Y;\ s : X \leftrightarrow Z \bullet \\ \quad\quad \{\ x : X;\ p : Y \times Z \mid x \mapsto p.1 \in r \land x \mapsto p.2 \in s\ \}\end{array}$$

split:

$$\text{split}[X, Y, Z] == \lambda\, r : X \leftrightarrow Y \times Z \bullet (r \fatsemi \text{first},\ r \fatsemi \text{second})$$

Examples

1. $r = \{x \mapsto y, x' \mapsto y', x'' \mapsto y''\}$
   $s = \{x \mapsto z, x' \mapsto z', x'' \mapsto z''\}$
   $\#\{x, x', x''\} = 3$
   $\text{merge}(r, s) = \{x \mapsto (y, z), x \mapsto (y', z'), x'' \mapsto (y'', z'')\}$
2. $r = \{z_1 \mapsto (x_1, y_1), \ldots, z_n \mapsto (x_n, y_n)\}$
   $\text{split}\, r = (\{z_1 \mapsto x_1, \ldots, z_n \mapsto x_n\}, \{z_1 \mapsto y_1, \ldots, z_n \mapsto y_n\})$

Laws

Law 20.44 *merge* is a total surjection (§21).

$$[X, Y, Z] \vdash \text{merge}[X, Y, Z] \in (X \leftrightarrow Y) \times (X \leftrightarrow Z) \twoheadrightarrow (X \leftrightarrow Y \times Z)$$

Law 20.45 An alternative definition of composition:

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z \vdash r \fatsemi s = s \circ r = \text{ran}(\text{merge}(r^\sim, s))$$

Law 20.46 *split* is a total injection (§21). Its range is the set of all pairs of relations with the same domain.

$$[X, Y, Z] \vdash \mathrm{split}[X, Y, Z] \in (X \leftrightarrow Y \times Z) \rightarrowtail\!\!\!\!\!\twoheadrightarrow \{\ r : X \leftrightarrow Y;\ s : X \leftrightarrow Z \mid \mathrm{dom}\, r = \mathrm{dom}\, s\ \}$$

Law 20.47 Splitting then merging restores the original relation.

$$[X, Y, Z] \vdash \mathrm{merge} \circ \mathrm{split} = \mathrm{id}(X \leftrightarrow Y \times Z)$$

Law 20.48 Merging then splitting restores the original relations on their common domain.

$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : X \leftrightarrow Z \vdash (\mathrm{split} \circ \mathrm{merge})(r, s) = (\mathrm{dom}\ \ s \lhd r, \mathrm{dom}\ \ r \lhd s)$$

•

Bicomposition

Intent

Given a function from pairs of arguments to a result $f : A \times B \nrightarrow C$, we can form a related function from pairs of sequences to a corresponding sequence of results $fb : \mathrm{seq}\, A \times \mathrm{seq}\, B \nrightarrow \mathrm{seq}\, C$. *bicompose* generalises this idea from functions to relations, and from sequences to labelled sets.

Definition

$$\begin{array}{l} \mathrm{bicompose}[A, B, C, X] == \\ \quad \lambda\, r : A \times B \leftrightarrow C \bullet \\ \quad\quad \lambda\, sa : X \leftrightarrow A;\ sb : X \leftrightarrow B \bullet \\ \quad\quad\quad \{\ x : X;\ c : C \mid (\ \exists\, a : A;\ b : B \bullet \\ \quad\quad\quad\quad x \mapsto a \in sa \land x \mapsto b \in sb \land (a, b) \mapsto c \in r\ )\ \} \end{array}$$

Examples

1. To apply a function $f : \mathbb{R}^{2\times} \to \mathbb{R}$ to the frequencies of a pair of bags, b and c, we can write $\text{bicompose}\, f(\text{count}\, b, \text{count}\, c) \rhd \{0\}$. For example,

$$\forall\, b, c : \text{bag}\, X \bullet b \uplus c = \text{bicompose}(\_ + \_)(\text{count}\, b, \text{count}\, c) \rhd \{0\}$$

Laws

Law 20.49 An alternative definition of bicomposition:

$$[A, B, C, X]\ r : A \times B \leftrightarrow C;\ sa : X \leftrightarrow A;\ sb : X \leftrightarrow B \vdash \\ \text{bicompose}\, r(sa, sb) = r \circ \text{merge}(sa, sb)$$

Law 20.50 Although bicomposition is defined in terms of any relations, it is motivated by the case where the relations are functions.

$$[A, B, C, X]\ f : A \times B \to C;\ g : X \to A;\ h : X \to B;\ x : X \vdash \\ \text{bicompose}\, f(g, h)x = f(g\ x, h\ x)$$

Bicomposition is useful where for example g and h are both sequences of the same length, or both bags over the same set.

•

Chapter 21

Functions

A function is an association of argument with a result, so that each argument produces (at most) one result. Different formal systems have differing approaches to the model of a function. For example, programming languages usually equate a function with an algorithm to proceed from argument to result, while the λ-calculus puts the text of the function in front of the text of the argument and provides rewriting rules to simplify if possible.

The set-theoretic mathematical approach to a function is to consider the argument and its result not as an algorithm or process, but rather as a possible pair. The function is identified with the set of all possible argument-result pairs. So in Z we can assimilate functions as special cases of binary relations, since both are just sets of pairs. We (arbitrarily) put the argument as the domain element and the result as the range element, and then consider as a function any relation where for each domain value there is at most one corresponding range value in the relation.

The advantage of identifying functions as special cases of relations is that definitions useful for work with relations are automatically inherited by functions. The disadvantage, if one is not careful, is that one may confuse ideas that ought to be kept separate.

- functional at a point: $(\_ \,\mathrm{functionalAt}\, \_)$
- (partial) functions: $X \pfun Y$
- total functions: $X \fun Y$
- finite functions: $X \ffun Y$
- surjections, total surjections, finite surjections: $X \psurj Y$, $X \surj Y$, $X \ffunsurj Y$
- injections, total injections, finite injections: $X \pinj Y$, $X \inj Y$, $X \finj Y$
- surjective injections, bijections, finite surjective injections: $X \pfunsurj Y$, $X \bij Y$, $X \ffunsurj Y$
- homomorphisms

- isomorphisms

Functionality

Intent

For any source set X, and target set Y, we can declare

$$f : X \nrightarrow Y$$

and f is a function provided there is a unique range element corresponding to each domain element:

$$\forall\, x : \mathrm{dom}\, f \bullet \exists_1\, y : Y \bullet x \mapsto y \in f$$

If we have some particular domain value $x : X$ the result of applying the function f is

$$(\ \mu\, y : Y \mid x \mapsto y \in f\ )$$

and we can write this as $f\, x$ (adding parentheses around the function, or the argument, or both together, if this assists clarity). We can also use the form $r\, x$, where r is a more general relation, not a function, provided the corresponding μ-term has a unique value, that is, when the relation is 'functional at the point of application'.

Definition

$$\mathrm{relation}\,(\_\,\mathrm{functionalAt}\,\_)$$
$$\_\,\mathrm{functionalAt}\,\_\,[X, Y] == \{\ r : X \leftrightarrow Y;\ x : X \mid \exists_1\, y : Y \bullet x \mapsto y \in r\ \}$$

A relation is functional at the point of application of an argument in its source set if there is a unique value that this maps to in the target set.

Examples

1. $\{x \mapsto y\}\,\mathrm{functionalAt}\, x$
2. $(r \oplus \{x \mapsto y\})\,\mathrm{functionalAt}\, x$
3. $root == \{\ i : \mathbb{Z} \bullet i * i \mapsto i\ \}$
   $root\,\mathrm{functionalAt}\, 0$
   $\neg\ (root\,\mathrm{functionalAt}\, 4)$, because $\{4 \mapsto 2, 4 \mapsto -2\} \subseteq root$
   $\neg\ (root\,\mathrm{functionalAt}\, 3)$, because $3 \notin \mathrm{dom}\, root$

Laws

Law 21.1 Function application syntax can be used to denote the unique target element corresponding to a particular domain element. This notation can also be used for relations, provided the relation is functional at the point of application.

$$[X, Y]\ r : X \leftrightarrow Y;\ x : X;\ y : Y \mid x \mapsto y \in r \land r \,\mathrm{functionalAt}\, x \vdash r\ x = y$$

Law 21.2 If r is functional at x, then the upper image through r of the set containing x is the singleton set containing $r\ x$. (This law explains the choice of notation $\_(\!|\ \_\ |\!)$ for upper image.)

$$[X, Y]\ r : X \leftrightarrow Y;\ x : X \mid r \,\mathrm{functionalAt}\, x \vdash r(\!|\ \{x\}\ |\!) = \{r(x)\}$$

Law 21.3 r is a function precisely when it is functional everywhere on its domain.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r \in X \nrightarrow Y \Leftrightarrow (\ \forall x : \mathrm{dom}\, r \bullet r \,\mathrm{functionalAt}\, x\ )$$

•

General functions

Intent

A function is a binary relation where each element in the domain maps to precisely one element in the range. A function is sometimes called a *many-one* relation.

Definition

(partial) functions:

$$\mathrm{generic}\ 5\,\mathrm{rightassoc}\,(\_ \nrightarrow \_)$$
$$X \nrightarrow Y == \{\ f : X \leftrightarrow Y \mid (\ \forall p, q : f \mid p.1 = q.1 \bullet p = q\ )\ \}$$

Examples

1. figure 21.1a shows a diagram of a partial function
2. a *tree* is a partial function from child node to parent node
3. sequences (§34) are functions from an initial segment of the natural numbers

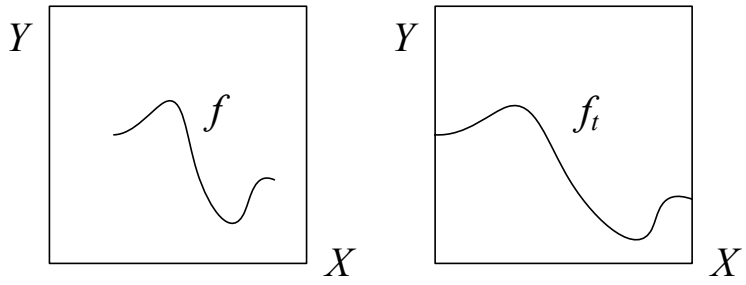


Figure 21.1 an example of (a) a partial function (b) a total function

---

Laws

Law 21.4 Two functions are equal precisely when they have equal domains and are compatible.

$$[X, Y]\ f, g : X \pfun Y \vdash f = g \Leftrightarrow \operatorname{dom} f = \operatorname{dom} g \wedge f \approx g$$

■ **Law 21.5** The function constructor distributes through non-empty generalised intersection.

$$[X, Y]\ \alpha : \mathbb{P}_1 \mathbb{P}\, X;\ \beta : \mathbb{P}_1 \mathbb{P}\, Y \vdash \bigcap \alpha \pfun \bigcap \beta = \bigcap \{\ a : \alpha;\ b : \beta \bullet a \pfun b\ \}$$
$$[X, Y]\ a, a' : \mathbb{P}\, X \vdash (a \cap a') \pfun Y = (a \pfun Y) \cap (a' \pfun Y)$$
$$[X, Y]\ b, b' : \mathbb{P}\, Y \vdash X \pfun (b \cap b') = (X \pfun b) \cap (X \pfun b')$$

Law 21.6 As a corollary of law 26.37 specialised to subsets, along with law 21.5, the generic partial function constructor is subset order-preserving on both its arguments.

$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash a \pfun b \subseteq X \pfun Y$$

Law 21.7 Any subset of a function is a function. In particular, a restricted function is a function, and the intersection of a function and a relation is a function.

$$[X, Y]\ f : X \pfun Y;\ r : X \rel Y \mid r \subseteq f \vdash r \in X \pfun Y$$
$$[X, Y]\ a : \mathbb{P}\, X;\ f : X \pfun Y;\ b : \mathbb{P}\, Y \vdash a \lhd f \rhd b \in X \pfun Y$$
$$[X, Y]\ f : X \pfun Y;\ r : X \rel Y \vdash f \cap r \in X \pfun Y$$

Law 21.8 The union of two functions is a function precisely when they are compatible.

$$[X, Y]\ f, g : X \pfun Y \vdash f \approx g \Leftrightarrow f \cup g \in X \pfun Y$$

Law 21.9 Overriding a function with a function yields a function.

$$[X, Y]\ f, g : X \pfun Y \vdash f \oplus g \in X \pfun Y$$

Law 21.10 Composing a function with a function yields a function.

$$[X, Y, Z]\ f : X \pfun Y;\ g : Y \pfun Z \vdash f \fatsemi g \in X \pfun Z$$

■ **Law 21.11** A relation is a function precisely when the composition of its inverse with itself yields the identity.

$$[X, Y]\ r : X \rel Y \vdash r^\sim \fatsemi r = \mathrm{id}(\mathrm{ran}\, r) \Leftrightarrow r \in X \pfun Y$$

Law 21.12 If two relations compose to give the identity, then on their composed parts they are inverses to each other and the second is a function,

$$[X, Y]\ r : X \rel Y;\ s : Y \rel X;\ a : \mathbb{P}\, X \mid r \fatsemi s = \mathrm{id}\, a \vdash \\ \exists\, b == \mathrm{ran}\, r \cap \mathrm{dom}\, s \bullet (r \rhd b)^\sim = b \lhd s \in b \rightarrowtail\!\!\!\!\!\!\rightarrow a$$

———————————————•

Total functions

Intent

If the domain of the function is the entire source set X, the function is a *total* function; function application is defined for any element of the source set.

Definition

total functions:

$$\mathrm{generic}\ 5\ \mathrm{rightassoc}\ (\_ \fun \_)$$
$$X \fun Y == \{\ f : X \pfun Y \mid \mathrm{dom}\, f = X\ \}$$

The use of the word ‘partial’ in partial function is not necessary; *all* functions are partial. It is used if needed to draw attention to the absence of the word ‘total’.

Examples

1. figure 21.1b shows a diagram of a total function
2. $square == \lambda\, i : \mathbb{Z} \bullet i * i$ is a total function on the integers: $square \in \mathbb{Z} \to \mathbb{Z}$
3. cosine is a total function on the reals: $\mathbb{R} \lhd \cos \in \mathbb{R} \to \mathbb{R}$

Laws

Law 21.13 Any function is total on its domain. (The phrase 'a function f is total on a' means '$a \subseteq \operatorname{dom} f$'.)

$$[X, Y]\ f : X \nrightarrow Y \vdash f \in \operatorname{dom} f \to Y$$

■ **Law 21.14** If two source sets are unequal, the corresponding sets of total functions are disjoint.

$$[X, Y]\ a, a' : \mathbb{P}\, X \mid \#\{a, a'\} = 2 \vdash \operatorname{disjoint}\langle a \to Y, a' \to Y\rangle$$

■ **Law 21.15** The total function constructor distributes through non-empty generalised intersection

$$[X, Y]\ \beta : \mathbb{P}_1\, \mathbb{P}\, Y \vdash X \to \bigcap \beta = \bigcap\{\ b : \beta \bullet X \to b\ \}$$
$$[X, Y]\ b, b' : \mathbb{P}\, Y \vdash X \to (b \cap b') = (X \to b) \cap (X \to b')$$

Law 21.16 As a corollary of law 26.37 specialised to subsets, along with law 21.15, the generic total function constructor is subset order-preserving on its target set argument.

$$[X, Y]\ b : \mathbb{P}\, Y \vdash X \to b \subseteq X \to Y$$

Law 21.17 Overriding a total function with a function yields a total function.

$$[X, Y]\ f : X \to Y;\ g : X \nrightarrow Y \vdash f \oplus g \in X \to Y$$

Law 21.18 Composing a total function with a function yields a total function precisely when the domain of the second includes the range of the first.

$$[X, Y, Z]\ f : X \to Y;\ g : Y \nrightarrow Z \vdash f \fatsemi g \in X \to Z \Leftrightarrow \operatorname{ran} f \subseteq \operatorname{dom} g$$

•

Finite functions

Intent

Define explicitly finite functions.

Definition

finite functions:

$$\text{generic } 5 \text{ rightassoc } (\_ \pfun \_)$$
$$X \pfun Y == (\ X \fun Y\ ) \cap (\ X \finj Y\ )$$

Laws

Law 21.19 A function is finite precisely when its domain is finite

$$[X, Y, Z]\ f : X \fun Y \vdash \text{finite}\, f \Leftrightarrow \text{finite}\, \text{dom}\, f$$

Surjections

Intent

If the range of the function is the entire target set Y, the function is a surjective function, or *surjection*; any element of the target set can be obtained by applying the function to some argument. A surjection is sometimes called an 'onto' function.

Definition

(partial) surjections:

$$\text{generic } 5 \text{ rightassoc } (\_ \psurj \_)$$
$$X \psurj Y == \{\ f : X \fun Y \mid \text{ran}\, f = Y\ \}$$

total surjections:

If the domain of the function is the entire source set X, and the range of the function is the entire target set Y, the function is a total surjection.

$$\text{generic } 5 \text{ rightassoc } (\_ \twoheadrightarrow \_)$$
$$X \twoheadrightarrow Y == (X \nrightarrow Y) \cap (X \rightarrow Y)$$

finite surjections:

$$\text{generic } 5 \text{ rightassoc } (\_ \twoheadrightarrow\!\!\!\!\!\!\rightarrow \_)$$
$$X \twoheadrightarrow\!\!\!\!\!\!\rightarrow Y == (X \nrightarrow Y) \cap (X \twoheadrightarrow\!\!\!\!\!\!\rightarrow Y)$$

Examples

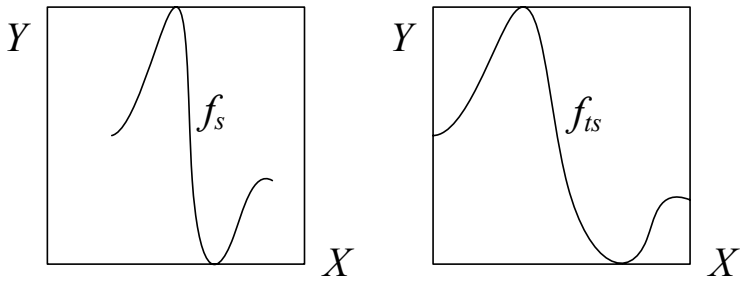


Figure 21.2 an example of (a) a partial surjection (b) a total surjection

1. figure 21.2 shows a diagram of a partial surjection and of a total surjection
2. addition on the integers is a total surjection: $\mathbb{Z}^{2\times} \lhd (\_ + \_) \in \mathbb{Z}^{2\times} \twoheadrightarrow \mathbb{Z}$
3. $ihalve == \lambda\, i : \mathbb{Z} \bullet i \operatorname{div} 2$ is a total surjection: $ihalve \in \mathbb{Z} \twoheadrightarrow \mathbb{Z}$
4. the binary operator of a *Monoid* (§23) is a surjection

Laws

Law 21.20 Any function is surjective onto its range. (The phrase 'a function f is surjective onto b' means '$b \subseteq \operatorname{ran} f$'.)

$$[X, Y]\ f : X \nrightarrow Y \vdash f \in X \twoheadrightarrow\!\!\!\!\!\!\rightarrow \operatorname{ran} f$$

Law 21.21 The surjection constructor distributes through non-empty generalised intersection

$$[X, Y]\ \alpha : \mathbb{P}_1\, \mathbb{P}\, X \vdash \bigcap \alpha \twoheadrightarrow Y = \bigcap \{\ a : \alpha \bullet a \twoheadrightarrow Y\ \}$$
$$[X, Y]\ a, a' : \mathbb{P}\, X \vdash (a \cap a') \twoheadrightarrow Y = (a \twoheadrightarrow Y) \cap (a' \twoheadrightarrow Y)$$

Law 21.22 As a corollary of law 26.37 specialised to subsets, along with law 21.21, the generic surjection constructor is subset order-preserving on its source set argument.

$$[X, Y]\ a : \mathbb{P}\, X \vdash a \twoheadrightarrow Y \subseteq X \twoheadrightarrow Y$$

Law 21.23 If two target sets are unequal, the corresponding sets of surjections are disjoint.

$$[X, Y]\ b, b' : \mathbb{P}\, Y \mid \#\{b, b'\} = 2 \vdash \mathrm{disjoint}\langle X \twoheadrightarrow b, X \twoheadrightarrow b'\rangle$$

Law 21.24 A function f is a surjection precisely when every two distinct functions give distinct post-compositions with f. (The post-compositions need not be distinct in the non-surjective case, if the differences lie outside the range of f.)

$$[X, Y, Z]\ f : X \nrightarrow Y \vdash$$
$$f \in X \twoheadrightarrow Y \Leftrightarrow (\ \forall g, h : Y \nrightarrow Z \mid f \fatsemi g = f \fatsemi h \bullet g = h\ )$$

Law 21.25 Overriding a surjection with a function covering the same range yields a surjection.

$$[X, Y]\ f : X \twoheadrightarrow Y;\ g : X \nrightarrow Y \mid \mathrm{ran}\, g = \mathrm{ran}(\mathrm{dom}\, g \lhd f) \vdash f \oplus g \in X \twoheadrightarrow Y$$

■ **Law 21.26** There are no surjections from a set onto its power set. The set $\mathbb{P}\, X$ is 'bigger' than the set X.

$$[X]\ \vdash X \twoheadrightarrow \mathbb{P}\, X = \varnothing$$

If X is a finite set with precisely N elements, then $\mathbb{P}\, X$ has 2^N elements. But there can be no surjection from a set with N elements onto a set with 2^N elements, since the latter has too many elements.

As X gets bigger, this clearly gets worse for all finite X. The theorem shows that taking X as infinite does not solve the problem, but rather that there must be in some sense 'different sizes' of infinite sets, with $\mathbb{P}\, X$ always bigger than X.

———————————————•

Injections

Intent

If each element in the domain maps to a different element in the range, the function is an injective function, or *injection*. An injection is sometimes called a *one-one* function.

Definition

(partial) injections:

$$\text{generic } 5 \text{ rightassoc } (\_ \pfun{1} \_)$$
$$X \pfun{1} Y == \{ f : X \pfun Y \mid (\forall p, q : f \bullet p.1 = q.1 \Leftrightarrow p.2 = q.2) \}$$

total injections:

$$\text{generic } 5 \text{ rightassoc } (\_ \fun{1} \_)$$
$$X \fun{1} Y == (X \pfun{1} Y) \cap (X \fun Y)$$

finite injections:

$$\text{generic } 5 \text{ rightassoc } (\_ \finj \_)$$
$$X \finj Y == (X \pfun Y) \cap (X \pfun{1} Y)$$

(partial) surjective injections:

$$\text{generic } 5 \text{ rightassoc } (\_ \psurj{1} \_)$$
$$X \psurj{1} Y == (X \psurj Y) \cap (X \pfun{1} Y)$$

bijections:

A function that is a total surjective injection is called a bijection.

$$\text{generic } 5 \text{ rightassoc } (\_ \bij \_)$$
$$X \bij Y == (X \psurj Y) \cap (X \fun{1} Y)$$

finite surjective injections:

$$\text{generic } 5 \text{ rightassoc } (\_ \fbij \_)$$
$$X \fbij Y == (X \finj Y) \cap (X \psurj{1} Y)$$

Instead of declaring a function f to be a finite surjective injection, $X \rightarrowtail\!\!\!\!\!\twoheadrightarrow Y$, we could instead declare the inverse of f to be a total injection, $Y \rightarrowtail X$, where additionally we have that Y is finite, *finite* Y.

Examples

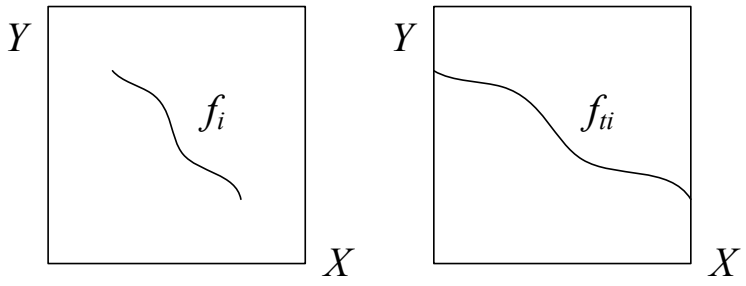


Figure 21.3 an example of (a) a partial injection (b) a total injection

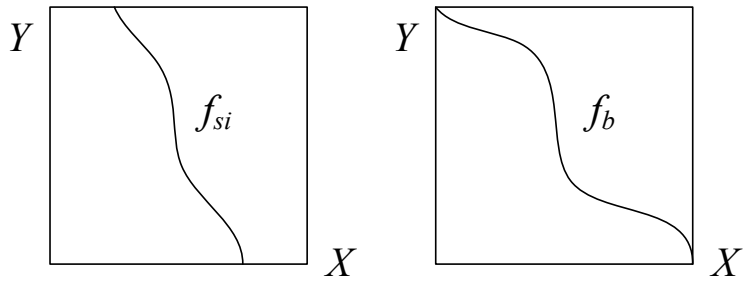


Figure 21.4 an example of (a) a surjective injection (b) a bijection

1. Figure 21.3 shows a diagram of a partial injection and of a total injection.
2. A total injection can be used to 'relabel' all the elements of X as elements of Y. There may be elements of Y 'left over'.
3. The Cartesian square $\_^{2\times}$ (§12) is a total injection.
4. Figure 21.4 shows a diagram of a surjective injection and of a bijection.
5. $succ == \lambda\, n : \mathbb{N} \bullet n + 1$, the successor function that adds one, is a total injection: $succ \in \mathbb{N} \rightarrowtail \mathbb{N}$. It is not surjective onto the naturals, because zero is not in its range. It is surjective onto the non-zero naturals, however: $succ \in \mathbb{N} \rightarrowtail\!\!\!\!\!\twoheadrightarrow \mathbb{N}_1$.

6. $pred == \lambda\, n : \mathbb{N}_1 \bullet n - 1$, the predecessor function that subtracts one, is a surjective injection: $pred \in \mathbb{N} \rightarrowtail\!\!\!\!\!\twoheadrightarrow \mathbb{N}$. It is not total on the naturals, because zero is not in its domain. It is total on the non-zero naturals, however: $pred \in \mathbb{N}_1 \rightarrowtail\!\!\!\!\!\twoheadrightarrow \mathbb{N}$.
7. $halve == \{\ i : \mathbb{Z} \bullet 2 * i \mapsto i\ \}$ is a surjective injection on the integers: $halve \in \mathbb{Z} \rightarrowtail\!\!\!\!\!\twoheadrightarrow \mathbb{Z}$. It is not total on the integers, because no odd numbers are in its domain.
8. Injective sequences (§34) are so called because they are injections.
9. A bijection can be used to 'relabel' all the elements of X as elements of Y, and *vice versa*.
10. Relational inverse is a bijection.
11. Unary negation of $\mathbb{R}$ is a bijection.
12. Sequence reversal on finite sequences is a bijection :
$\operatorname{seq} X \lhd \operatorname{rev} \in \operatorname{seq} X \rightarrowtail\!\!\!\!\!\twoheadrightarrow \operatorname{seq} X$

Laws

Law 21.27 The injection constructor distributes through non-empty generalised intersection

$$[X, Y]\ \alpha : \mathbb{P}_1 \mathbb{P}\, X;\ \beta : \mathbb{P}_1 \mathbb{P}\, Y \vdash \bigcap \alpha \rightarrowtail \bigcap \beta = \bigcap \{\ a : \alpha;\ b : \beta \bullet a \rightarrowtail b\ \}$$
$$[X, Y]\ a, a' : \mathbb{P}\, X \vdash (a \cap a') \rightarrowtail Y = (a \rightarrowtail Y) \cap (a' \rightarrowtail Y)$$
$$[X, Y]\ b, b' : \mathbb{P}\, Y \vdash X \rightarrowtail (b \cap b') = (X \rightarrowtail b) \cap (X \rightarrowtail b')$$

Law 21.28 As a corollary of law 26.37 specialised to subsets, along with law 21.27, the generic injection constructor is subset order-preserving on both its arguments.

$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash a \rightarrowtail b \subseteq X \rightarrowtail Y$$

Law 21.29 Any subset of an injection is an injection. In particular, a restricted injection is an injection, and the intersection of an injection and a relation is a injection.

$$[X, Y]\ f : X \rightarrowtail Y;\ r : X \leftrightarrow Y \mid r \subseteq f \vdash r \in X \rightarrowtail Y$$
$$[X, Y]\ a : \mathbb{P}\, X;\ f : X \rightarrowtail Y;\ b : \mathbb{P}\, Y \vdash a \lhd f \rhd b \in X \rightarrowtail Y$$
$$[X, Y]\ f : X \rightarrowtail Y;\ r : X \leftrightarrow Y \vdash f \cap r \in X \rightarrowtail Y$$

Law 21.30 The union of two injections is an injection precisely when they are compatible and their inverses are compatible.

$$[X, Y]\ f, g : X \rightarrowtail Y \vdash f \approx g \land f^{\sim} \approx g^{\sim} \Leftrightarrow f \cup g \in X \rightarrowtail Y$$

Law 21.31 The inverse of an injection is an injection.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r \in X \rightarrowtail Y \Leftrightarrow r^{\sim} \in Y \rightarrowtail X$$

Law 21.32 Overriding an injection with an injection covering the same range yields an injection.

$$[X, Y]\ f, g : X \rightarrowtail Y \mid \operatorname{ran} g = \operatorname{ran}(\operatorname{dom} g \lhd f) \vdash f \oplus g \in X \rightarrowtail Y$$

Law 21.33 As an immediate corollary of law 21.11, a function is an injection precisely when its composition with its inverse yields the identity.

$$[X, Y]\ f : X \nrightarrow Y \vdash f \fatsemi f^{\sim} = \operatorname{id}(\operatorname{dom} f) \Leftrightarrow f \in X \rightarrowtail Y$$

Law 21.34 A relation is a total injection from some domain precisely when its inverse is a surjective injection onto that range.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r \in X \rightarrowtail Y \Leftrightarrow r^{\sim} \in Y \twoheadrightarrow X$$

This law shows that if a surjective injection arises in a specification, it is possible to work instead with a total injection, its inverse. This may be stylistically preferable, since a total injection may be the easier concept to understand.

Law 21.35 A function f is an injection precisely when every two distinct functions with the same range that is a subset of f's domain give distinct pre-compositions with f. (The reason that pre-compositions need not be distinct when f is not an injection is that the non-injective f can merge, or 'confuse', different elements.)

$$\begin{array}{l}[X, Y, Z]\ f : X \nrightarrow Y \vdash \\ \quad f \in X \rightarrowtail Y \Leftrightarrow \\ \qquad (\ \forall g, h : Z \nrightarrow X \mid \operatorname{ran} g = \operatorname{ran} h \subseteq \operatorname{dom} f \land g \fatsemi f = h \fatsemi f \bullet g = h\ )\end{array}$$

Law 21.36 The composition of injections is an injection.

$$[X, Y, Z]\ f : X \rightarrowtail Y;\ g : Y \rightarrowtail Z \vdash f \fatsemi g \in X \rightarrowtail Z$$

Law 21.37 Consider two functions whose composition is an injection. If the range of the first function is a subset of the domain of the second, then the first is an injection. If the range and domain are in fact equal, then the second is also an injection.

$$[X, Y, Z]\ f : X \pfun Y;\ g : Y \pfun Z \mid f \fatsemi g \in X \rightarrowtail Z \land \operatorname{ran} f \subseteq \operatorname{dom} g \vdash f \in X \rightarrowtail Y$$

$$[X, Y, Z]\ f : X \pfun Y;\ g : Y \pfun Z \mid f \fatsemi g \in X \rightarrowtail Z \land \operatorname{ran} f = \operatorname{dom} g \vdash f \in X \rightarrowtail Y \land g \in Y \rightarrowtail Z$$

Law 21.38 Compositions can be cancelled on the right when they are total injections.

$$[X, Y, Z]\ r, s : X \rightarrow Y;\ a : \mathbb{P}\, Y;\ f : Y \rightarrowtail Z \mid \operatorname{ran} r \cup \operatorname{ran} s \subseteq a \land f \in a \rightarrowtail Z \vdash r \fatsemi f = s \fatsemi f \Leftrightarrow r = s$$

Law 21.39 Compositions can be cancelled on the left when they are surjective injections.

$$[X, Y, Z]\ f : X \rightarrowtail Y;\ a : \mathbb{P}\, Y;\ r, s : Y \rightarrow Z \mid f \in X \twoheadrightarrow a \land \operatorname{dom} r \cup \operatorname{dom} s \subseteq a \vdash f \fatsemi r = f \fatsemi s \Leftrightarrow r = s$$

Law 21.40 There are no total injections from a power set to its set.

$$[X]\ \vdash \mathbb{P}\, X \rightarrowtail X = \varnothing$$

Law 21.41 A function is a finite injection precisely when its range is finite

$$[X, Y, Z]\ f : X \rightarrowtail Y \vdash \operatorname{finite} f \Leftrightarrow \operatorname{finite} \operatorname{ran} f$$

Law 21.42 The inverse of a bijection is a bijection with domain and range interchanged.

$$[X, Y]\ f : X \rightarrowtail\!\!\!\!\!\rightarrow Y \vdash f^{\sim} \in Y \rightarrowtail\!\!\!\!\!\rightarrow X$$

Law 21.43 Composition with a bijection preserves the properties of a function.

$$[X, Y, Z]\ f : X \rightarrowtail\!\!\!\!\!\rightarrow Y;\ g : Y \rightarrow Z \vdash f \fatsemi g \in X \rightarrow Z$$
$$[X, Y, Z]\ f : X \rightarrowtail\!\!\!\!\!\rightarrow Y;\ g : Y \twoheadrightarrow Z \vdash f \fatsemi g \in X \twoheadrightarrow Z$$
$$[X, Y, Z]\ f : X \rightarrowtail\!\!\!\!\!\rightarrow Y;\ g : Y \rightarrowtail Z \vdash f \fatsemi g \in X \rightarrowtail Z$$
$$[X, Y, Z]\ g : X \rightarrow Y;\ f : Y \rightarrowtail\!\!\!\!\!\rightarrow Z \vdash g \fatsemi f \in X \rightarrow Z$$
$$[X, Y, Z]\ g : X \twoheadrightarrow Y;\ f : Y \rightarrowtail\!\!\!\!\!\rightarrow Z \vdash g \fatsemi f \in X \twoheadrightarrow Z$$
$$[X, Y, Z]\ g : X \rightarrowtail Y;\ f : Y \rightarrowtail\!\!\!\!\!\rightarrow Z \vdash g \fatsemi f \in X \rightarrowtail Z$$

Law 21.44 *Schröder-Bernstein theorem*: if there are total injections between two sets in both directions, then there is a bijection between them. (See [Halmos 1960], [Suppes 1972], [Enderton 1977].)

$$[X, Y]\ \vdash (\ \exists f : X \rightarrowtail Y;\ g : Y \rightarrowtail X \bullet \mathit{true}\ ) \Leftrightarrow (\ \exists h : X \rightarrowtail\!\!\!\!\!\rightarrow Y \bullet \mathit{true}\ )$$

———————————————•

Homomorphism

Intent

Structure-preserving maps are of interest. Here we define maps that preserve the structure of functions; in §26 we define maps that preserve the structure of order relations.

Consider a source set of elements a, and a binary homogeneous function $\diamond_x$ that is total on $a^{2\times}$. A target set b also has a total function $\diamond_y$ defined on it. A function h that maps elements of a to elements of b, whilst preserving the structure of the function $\diamond_x$, is a *homomorphism*.

Definition

function 30 leftassoc $(\_ \diamond_x \_)$

function 30 leftassoc $(\_ \diamond_y \_)$

Homomorphism $[X, Y]$

$a : \mathbb{P}\, X;\ \_ \diamond_x \_ : X^{2\times} \nrightarrow X$
$b : \mathbb{P}\, Y;\ \_ \diamond_y \_ : Y^{2\times} \nrightarrow Y$
$h : X \nrightarrow Y$

$a^{2\times} \lhd (\_ \diamond_x \_) \in a^{2\times} \to a$
$b^{2\times} \lhd (\_ \diamond_y \_) \in b^{2\times} \to b$
$a \lhd h \in a \to b$
$\forall\, x, y : a \bullet h(x \diamond_x y) = h\ x \diamond_y h\ y$

Examples

1. The modulus operation forms a homomorphism from addition, to addition modulo n.

$$
\begin{array}{l}
n : \mathbb{N}_1 \vdash \\
\quad \exists\, \mathrm{Homomorphism}[\mathbb{A}, \mathbb{A}] \bullet \\
\qquad a = \mathbb{N} \land (\_ \diamond_x \_) = (\_ + \_) \\
\qquad \land\ b = 0 \,..\, n-1 \land (\_ \diamond_y \_) = (\_ + \_) \bmod n \\
\qquad \land\ h = (\ \lambda\, m : \mathbb{N} \bullet m \bmod n\ )
\end{array}
$$

2. When the two binary operators are the same (except possibly for different generic instantiations), then a homomorphism becomes the statement of a distributive law: h distributes through the binary operator
$h(x \diamond y) = (h\ x) \diamond (h\ y)$

3. Upper image forms a homomorphism on set union:

$$
\begin{array}{l}
[X, Y]\ r : X \leftrightarrow Y \vdash \\
\quad \exists\, \mathrm{Homomorphism}[\mathbb{P}\, X, \mathbb{P}\, Y] \bullet \\
\qquad a = \mathbb{P}\, X \land (\_ \diamond_x \_) = (\_ \cup \_) \\
\qquad \land\ b = \mathbb{P}\, Y \land (\_ \diamond_y \_) = (\_ \cup \_) \\
\qquad \land\ h = (\ \lambda\, c : \mathbb{P}\, X \bullet \mathrm{upperImage}\, r\ c\ )
\end{array}
$$

   This homomorphism is expressed as the distributive law 19.4.

4. When the two binary operators are set union and intersection (or some other similarly 'paired' operations), then a homomorphism becomes the statement of a pseudo-distributive law: h pseudo-distributes through union (or intersection)
$h(x \cup y) = (h\ x) \cap (h\ y)$

5. Domain subtraction creates a homomorphism between set union and intersection:

$$
\begin{array}{l}
[X, Y]\ r : X \leftrightarrow Y \vdash \\
\quad \exists\, \mathrm{Homomorphism}[\mathbb{P}\, X, X \leftrightarrow Y] \bullet \\
\qquad a = \mathbb{P}\, X \land (\_ \diamond_x \_) = (\_ \cap \_) \\
\qquad \land\ b = X \leftrightarrow Y \land (\_ \diamond_y \_) = (\_ \cup \_) \\
\qquad \land\ h = (\ \lambda\, c : \mathbb{P}\, X \bullet c \lhd r\ )
\end{array}
$$

   This homomorphism is expressed as the pseudo-distributive laws 18.8 and 18.9.

6. A requirement on a safety critical compiler might be that it should create a homomorphism between statement composition and instruction concatenation, in order that the target code be more readily verifiable:

$$
\begin{array}{l}
\mathcal{C} : STMT \nrightarrow \mathrm{seq}\, INSTR \\
\mathcal{C}(s \fatsemi s') = \mathcal{C}\ s \frown \mathcal{C}\ s'
\end{array}
$$

   An optimising compiler, on the other hand, would not in general create such a homomorphism.

Isomorphism

Intent

If the homomorphism's structure-preserving map is bijective, it is called an *isomorphism*.

In such a case the sets a and b are *isomorphic* with respect to the operations, and can be considered 'identical up to a renaming'. This concept is particularly useful in a typed language like Z, to relate two isomorphic sets over different types, because we cannot say $a = b$ and be type correct.

Definition

Isomorphism $[X, Y]$

$\mathrm{Homomorphism}[X, Y]$

$a \lhd h \in a \rightarrowtail\!\!\!\!\!\rightarrow b$

Examples

1. The positive reals under multiplication are isomorphic to the reals under addition, where the isomorphism is the natural logarithm:

$$\begin{array}{l} \vdash \exists\, \mathrm{Isomorphism}[\mathbb{A}, \mathbb{A}] \bullet \\ \quad a = \mathbb{R}_+ \land (\_ \diamond_x \_) = (\_ * \_) \\ \quad \land\ b = \mathbb{R} \land (\_ \diamond_y \_) = (\_ + \_) \\ \quad \land\ h = (\ln \_) \end{array}$$

2. Cartesian product creates an isomorphism with respect to the four simple set operations

$$\begin{array}{l} [X,\, Y]\ y : \mathbb{P}\, Y \vdash \\ \quad \exists\, \mathrm{Isomorphism}[\mathbb{P}\, X, X \leftrightarrow Y] \bullet \\ \quad\quad a = \mathbb{P}\, X \land (\_ \diamond_x \_) = (\_ \cup \_) \\ \quad\quad \land\ b = X \leftrightarrow Y \land (\_ \diamond_y \_) = (\_ \cup \_) \\ \quad\quad \land\ h = (\ \lambda\, x : \mathbb{P}\, X \bullet x \times y\ ) \end{array}$$

$$\begin{array}{l} [X,\, Y]\ y : \mathbb{P}\, Y \vdash \\ \quad \exists\, \mathrm{Isomorphism}[\mathbb{P}\, X, X \leftrightarrow Y] \bullet \\ \quad\quad a = \mathbb{P}\, X \land (\_ \diamond_x \_) = (\_ \cap \_) \\ \quad\quad \land\ b = X \leftrightarrow Y \land (\_ \diamond_y \_) = (\_ \cap \_) \\ \quad\quad \land\ h = (\ \lambda\, x : \mathbb{P}\, X \bullet x \times y\ ) \end{array}$$

$$\begin{array}{l} [X,\, Y]\ y : \mathbb{P}\, Y \vdash \\ \quad \exists\, \mathrm{Isomorphism}[\mathbb{P}\, X, X \leftrightarrow Y] \bullet \\ \quad\quad a = \mathbb{P}\, X \land (\_ \diamond_x \_) = (\_ \setminus \_) \\ \quad\quad \land\ b = X \leftrightarrow Y \land (\_ \diamond_y \_) = (\_ \setminus \_) \\ \quad\quad \land\ h = (\ \lambda\, x : \mathbb{P}\, X \bullet x \times y\ ) \end{array}$$

$$\begin{array}{l} [X,\, Y]\ y : \mathbb{P}\, Y \vdash \\ \quad \exists\, \mathrm{Isomorphism}[\mathbb{P}\, X, X \leftrightarrow Y] \bullet \\ \quad\quad a = \mathbb{P}\, X \land (\_ \diamond_x \_) = (\_ \ominus \_) \\ \quad\quad \land\ b = X \leftrightarrow Y \land (\_ \diamond_y \_) = (\_ \ominus \_) \\ \quad\quad \land\ h = (\ \lambda\, x : \mathbb{P}\, X \bullet x \times y\ ) \end{array}$$

These isomorphisms are expressed as the distributive laws 9.19 and 9.20

3. Relational inverse creates an isomorphism with respect to the four simple set operations

$$
\begin{array}{l}
[X, Y] \vdash \\
\quad \exists\, \mathrm{Isomorphism}[X \leftrightarrow Y, Y \leftrightarrow X] \bullet \\
\qquad (\_ \diamond_x \_) = (\_ \cup \_) \wedge (\_ \diamond_y \_) = (\_ \cup \_) \wedge h = (\_^{\sim}) \\
[X, Y] \vdash \\
\quad \exists\, \mathrm{Isomorphism}[X \leftrightarrow Y, Y \leftrightarrow X] \bullet \\
\qquad (\_ \diamond_x \_) = (\_ \cap \_) \wedge (\_ \diamond_y \_) = (\_ \cap \_) \wedge h = (\_^{\sim}) \\
[X, Y] \vdash \\
\quad \exists\, \mathrm{Isomorphism}[X \leftrightarrow Y, Y \leftrightarrow X] \bullet \\
\qquad (\_ \diamond_x \_) = (\_ \setminus \_) \wedge (\_ \diamond_y \_) = (\_ \setminus \_) \wedge h = (\_^{\sim}) \\
[X, Y] \vdash \\
\quad \exists\, \mathrm{Isomorphism}[X \leftrightarrow Y, Y \leftrightarrow X] \bullet \\
\qquad (\_ \diamond_x \_) = (\_ \ominus \_) \wedge (\_ \diamond_y \_) = (\_ \ominus \_) \wedge h = (\_^{\sim})
\end{array}
$$

These isomorphisms are expressed as the distributive laws 16.7 and 16.9.

———————————————•

Chapter 22

Labelled Sets

A set is totally defined by its members, so that if any member of one set is a member of another, and vice versa, the two sets are identical. Any element is either a member of a set, or it is not. These simple properties make sets a suitable basis for our theory, but often the statement that some values form a set gives us insufficient information. As an example, it is easy to suppose naively that the set $\{a, b, c\}$ must have three members, but this is not so without the additional constraint that a, b, and c are all distinct. To specify that "all elements of the set are distinct" does not impose this constraint, since if any pair of a, b, or c is equal, the set will have only two members, so any two elements of it will still be distinct. One general way in which we can assert the distinctness of elements such as these is to pair them up with other values known to be distinct, then form a set of the pairs. Thus if $\{x, y, z\}$ are known to be distinct, and we form the labelled set $s = \{x \mapsto a, y \mapsto b, z \mapsto c\}$ then the predicate $\forall\, i, j : \mathrm{dom}\, s \mid i \neq j \bullet s\ i \neq s\ j$ will assert what we wanted to say.

A "labelled set" is therefore any function $L \nrightarrow \mathit{Value}$ where the range values are the sole focus of attention, and the purpose of the domain values is just to keep the range values apart. Numbers (§29) often form a convenient labelling set, and labelled sets often take the form of sequences (§34).

This understanding explains why the predicate "disjoint", defined below, has to use a labelled set of sets. If an unlabelled set of sets were used, different representations of the same set within the whole set would be treated as a single element, so the whole would be counted as disjoint.

- disjointness
- partition

Disjointness

Intent

If the values in a labelled set are themselves sets, they may have certain non-overlapping properties.

A collection of sets is *disjoint* if every pair of sets has no intersection.

Definition

relation (disjoint _)

$$\text{disjoint}\_\ [L, X] == \{\ r : L \leftrightarrow \mathbb{P}\, X \mid \forall\, p, q : r \mid p \neq q \bullet p.2 \cap q.2 = \varnothing\ \}$$

Examples

1. $\text{disjoint}\langle \mathit{prime}, \mathit{composite} \rangle$

Laws

Law 22.1 The empty set is disjoint; a singleton maplet set is disjoint; a two maplet set is disjoint precisely when the two labelled sets have no intersection.

$$[L, X]\ \vdash \text{disjoint}\, \varnothing[L \times \mathbb{P}\, X]$$
$$[L, X]\ l : L;\ a : \mathbb{P}\, X \vdash \text{disjoint}\{l \mapsto a\}$$
$$[L, X]\ k, l : L;\ a, b : \mathbb{P}\, X \mid \#\{k, l\} = 2 \vdash \text{disjoint}\{k \mapsto a, l \mapsto b\} \Leftrightarrow a \cap b = \varnothing$$

Law 22.2 If a family of sets is disjoint, their generalised intersection is empty.

$$[L, X]\ r : L \nrightarrow \mathbb{P}\, X \mid \text{disjoint}\, r \vdash \bigcap(\text{ran}\, r) = \varnothing$$

•

Partition

Intent

A collection of sets *partitions* a set a if the sets are disjoint and together form the whole of a.

Definition

$$\text{relation}\,(\_\,\text{partition}\,\_)$$
$$\_\,\text{partition}\,\_\,[L, X] == \{\ r : L \leftrightarrow \mathbb{P}\,X;\ a : \mathbb{P}\,X \mid \text{disjoint}\,r \land \bigcup(\text{ran}\,r) = a\ \}$$

Examples

1. $\langle\{0, 1\}, \mathit{prime}, \mathit{composite}\rangle\,\text{partition}\,\mathbb{N}$

Laws

Law 22.3 A pair of sets partition a third precisely when they are disjoint and together form the whole of the third set.

$$[L, X]\ l, l' : L;\ a, a', b : \mathbb{P}\,X \mid \#\{l, l'\} = 2 \vdash$$
$$\{l \mapsto a, l' \mapsto a'\}\,\text{partition}\,b \Leftrightarrow a \cap a' = \varnothing \land a \cup a' = b$$

Law 22.4 Any set can be partitioned in the following ways:

$$[X]\ a, b : \mathbb{P}\,X \vdash \langle a \cap b, a \setminus b\rangle\,\text{partition}\,a$$
$$[X]\ a, b : \mathbb{P}\,X \vdash \langle a \setminus b, b \setminus a\rangle\,\text{partition}(a \ominus b)$$
$$[X]\ a, b : \mathbb{P}\,X \vdash \langle a, b \setminus a\rangle\,\text{partition}(a \cup b)$$
$$[X]\ a, b : \mathbb{P}\,X \vdash \langle a \cap b, a \ominus b\rangle\,\text{partition}(a \cup b)$$

•

Chapter 23

Binary operators

In this chapter we explore several important properties of binary operators (functions that take two arguments).

- idempotent operators
- semigroups and abelian semigroups
- monoids and abelian monoids
- groups and abelian groups
- distribute Abelian monoid over finite labelled set
- finite distributed sum and product, +/, */

The set union operator $\cup$ has the property that $a \cup a = a$; it is *idempotent*.

The symmetric set difference operator $\ominus$ has the property that $a \ominus (b \ominus c)$ is the same as $(a \ominus b) \ominus c$; it is *associative*. It also has the property that there is an element $i = \varnothing$, such that for every a, $a \ominus i = i \ominus a = a$; it has an *identity element*. It also has the property that for any a, there is another element b, such that $a \ominus b = b \ominus a = i$; every element has an *inverse*. These properties of symmetric set difference are those of a well-known mathematical construct: a *group*.

More about groups and so on can be found in any book on 'abstract algebra', for example, [Bhattacharya *et al.* 1994], [Fraleigh 1994].

Many binary operators can be usefully generalised to a *distributed* form, where they are used to combine a collection of more than two values. For example, distributed addition can be used to sum a series, distributed concatenation can be used to combine many sequences.

Precisely how the distribution is defined depends on the properties of the binary operator. We assume that the binary operator is at least a monoid. Being associative we do not need to worry about bracketing. Having an identity element means

we can give a natural meaning to the case of combining an empty collection of values.

In this chapter we make the additional assumption that the operator is commutative (hence forms an abelian monoid). So we also do not have to worry about any order that the collection of values has. Later, after we have defined sequences (§34), we relax this assumption. We also require here that the collection of values is finite. This means that we do not have to worry about convergence properties in the general case. We relax this requirement in certain specific cases later, where appropriate.

If the operator is idempotent, we can put its arguments together in a set, since it does not matter how many times any particular argument has been provided. Thus for the application of union and intersection to many arguments, we use the $\bigcup$ and $\bigcap$ functions. For operators which are not idempotent, however, we need some way of distinguishing the arguments: we use some labelling function $L \twoheadrightarrow X$.

The general two-argument function is:

function 30 leftassoc $(\_ \diamond \_)$

Idempotence

Intent

An idempotent operator can be applied multiple times with no more effect than applying it once.

Definition

generic (idempotent _)

idempotent $X == \{ \_ \diamond \_ : X^{2\times} \to X \mid (\ \forall x : X \bullet x \diamond x = x \ ) \}$

Examples

1. $(\_ \cup \_)$, $(\_ \cap \_)$, and $(\_ \oplus \_)$ are idempotent.
2. If we want to specify an operation that has no further effect were we to 'mistakenly' perform it again, such as updating some record, or delivering some packet over a network, we can use an idempotent operator to construct our definition.

Semigroup

Intent

A *semigroup* consists of a set, and a two-argument function. On the set, the function is total (defined for all elements of the set), closed (mapping only to elements of the set), and associative.

A special case is when the operator is *abelian*: when it has the commutative property that $x \diamond y$ is the same as $y \diamond x$. (The name 'abelian' commemorates the contribution to group theory made by the Norwegian mathematician Niels Henrik Abel, 1802–1829.)

Definition

semigroups:

SemiGroup $[X]$

$$g : \mathbb{P}\, X$$
$$\_ \diamond \_ : X^{2\times} \nrightarrow X$$

$$g^{2\times} \lhd (\_ \diamond \_) \in g^{2\times} \rightarrow g$$
$$\forall\, x, y, z : g \bullet (x \diamond y) \diamond z = x \diamond (y \diamond z)$$

abelian semigroups:

$$\text{AbelianSemiGroup}[X] == [\ \text{SemiGroup}[X] \mid \forall\, x, y : g \bullet x \diamond y = y \diamond x\ ]$$

Examples

1. Payments into a bank account (without bank charges) should form an abelian semigroup: paying in two amounts should give the same balance as paying in a single amount equal to their sum, and the order the payments are made in should not affect the final balance.
2. Computer arithmetic does not form a semigroup, because it is not associative. Arithmetic overflow means that, on occasion, $x + (y + z) \neq (x + y) + z$. For

example, if $y \simeq -z$, so that $y + z \simeq 0$, the first expression might be well defined, but the second might overflow.

Variant

An alternative way of defining *SemiGroup* would be to describe

$$g^{2\times} \lhd (\_ \diamond \_)$$

alone as the semigroup. This is in a sense simpler, since it means that we would be working with a single relation, rather than with the two-component schema above. The schema approach turns out to have an easier development, however (see also the discussion in §3.7.2), and corresponds better with the long-established mathematical nomenclature. Some of the mathematical literature even goes so far as to describe the set g on its own as being a semigroup, leaving the operation implicit, but we shall not be so lax, nor so ambiguous.

---•

Monoid

Intent

An identity element leaves every other element unchanged by the operation. A monoid is a semigroup where one of the elements of the group set is an identity element.

Definition

monoids:

Monoid $[X]$

$$\text{SemiGroup}[X]$$
$$e : X$$

$$e \in g$$
$$\forall x : g \bullet x \diamond e = x = e \diamond x$$

abelian monoids:

$$\text{AbelianMonoid}[X] == \text{Monoid}[X] \land \text{AbelianSemiGroup}[X]$$

Examples

Once we have provided the set g and the operation $\diamond$ for a particular monoid, the identity element e is a *derived component* (uniquely fixed by the rest of the specification, law 23.1), and so need not be specified explicitly. We *do* specify it in these examples, however, to be helpful in identifying it to the reader. (Technically, we should *prove* the existence of the alleged monoid with its alleged identity element. We do not do so for these examples, because the proofs are unilluminating.)

1. Relational overriding forms a monoid over relations. The identity element is the empty set.

$$[X, Y]\ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash$$
$$\quad \exists\, \mathrm{Monoid}[X \leftrightarrow Y] \bullet g = a \leftrightarrow b \land (\_ \diamond \_) = (\_ \oplus \_) \land e = \varnothing$$

2. Homogeneous relational composition forms a monoid over relations, partial functions, total functions, surjections, and injections. The identity element is the identity relation.

$$[X]\ a : \mathbb{P}\, X \vdash$$
$$\quad \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \leftrightarrow a \land (\_ \diamond \_) = (\_ \fatsemi \_) \land e = \mathrm{id}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash$$
$$\quad \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \nrightarrow a \land (\_ \diamond \_) = (\_ \fatsemi \_) \land e = \mathrm{id}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash$$
$$\quad \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \rightarrow a \land (\_ \diamond \_) = (\_ \fatsemi \_) \land e = \mathrm{id}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash$$
$$\quad \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \twoheadrightarrow a \land (\_ \diamond \_) = (\_ \fatsemi \_) \land e = \mathrm{id}\, a$$
$$[X]\ a : \mathbb{P}\, X \vdash$$
$$\quad \exists\, \mathrm{Monoid}[X \leftrightarrow X] \bullet g = a \rightarrowtail a \land (\_ \diamond \_) = (\_ \fatsemi \_) \land e = \mathrm{id}\, a$$

3. Sequence concatenation forms a monoid over finite sequences. The identity element is the empty sequence.

$$[X]\ \vdash \exists\, \mathrm{Monoid}[\mathbb{A} \leftrightarrow X] \bullet g = \mathrm{seq}\, X \land (\_ \diamond \_) = (\_ \frown \_) \land e = \langle\, \rangle$$

4. Set union forms an abelian monoid over sets. The identity element is the empty set.

$$[X]\ \vdash \exists\, \mathrm{AbelianMonoid}[\mathbb{P}\, X] \bullet g = \mathbb{P}\, X \land (\_ \diamond \_) = (\_ \cup \_) \land e = \varnothing$$

5. Set intersection forms an abelian monoid over sets. The identity element is the full set.

$$[X] \vdash \exists\, \mathrm{AbelianMonoid}[\mathbb{P}\, X] \bullet g = \mathbb{P}\, X \land (\_ \diamond \_) = (\_ \cap \_) \land e = X$$

6. Arithmetic addition forms an abelian monoid over natural numbers. The identity element is zero.

$$\vdash \exists\, \mathrm{AbelianMonoid}[\mathbb{A}] \bullet g = \mathbb{N} \land (\_ \diamond \_) = (\_ + \_) \land e = 0$$

7. Arithmetic multiplication modulo n forms an abelian monoid over $0 \mathinner{..} n-1$. The identity element is unity.

$$n : \mathbb{N}_1 \vdash \exists\, \mathrm{AbelianMonoid}[\mathbb{A}] \bullet g = 0 \mathinner{..} n-1 \land (\_ \diamond \_) = (\_ * \_) \bmod n \land e = 1$$

Laws

■ **Law 23.1** A monoid has a unique identity element.

$$[X]\ \mathrm{Monoid}[X];\ e_0 : X \mid e_0 \in g \land (\ \forall\, x : g \bullet x \diamond e_0 = x\ ) \vdash e_0 = e$$
$$[X]\ \mathrm{Monoid}[X];\ e_0 : X \mid e_0 \in g \land (\ \forall\, x : g \bullet e_0 \diamond x = x\ ) \vdash e_0 = e$$

■ **Law 23.2** A monoid's binary function is a surjection.

$$[X]\ \mathrm{Monoid}[X] \vdash g^{2\times} \lhd (\_ \diamond \_) \in g^{2\times} \twoheadrightarrow g$$

---•

Group

Intent

An inverse element is one which combines with an element a to give the identity element. The inverse function returns the inverse of every element. A group is a monoid with an inverse function on the group's set.

Definition

groups:

$$\begin{array}{|l}\hline \text{Group}\,[X] \\ \text{Monoid}[X] \\ inv : X \nrightarrow X \\ \hline g \lhd inv \in g \rightarrow g \\ \forall\, x : g \bullet x \diamond inv\ x = inv\ x \diamond x = e \\ \hline \end{array}$$

abelian groups:

$$\text{AbelianGroup}[X] == \text{Group}[X] \land \text{AbelianSemiGroup}[X]$$

Examples

Once we have provided the set g and the operation $\diamond$ for a particular group, the identity element e and the inverse function inv are *derived components* (uniquely fixed by the rest of the specification, laws 23.1 and 23.3), and so need not be specified explicitly although we do so in the following examples. We *do* specify them in these examples, however, to be helpful in identifying them to the reader. (Technically, we should *prove* the existence of the alleged group with its alleged identity element and inverse function. We do not do so for these examples, because the proofs are unilluminating. We do so for the case of arithmetic modulo prime p, law 23.4, because the proof is not quite so immediately obvious.)

1. Relational composition forms a group over the set of homogeneous bijections. The identity element is the identity relation; the inverse function is relational inverse.

$$\begin{array}{l}[X] \vdash \\ \quad \exists\, \text{Group}[X \leftrightarrow X] \bullet \\ \qquad g = X \rightarrowtail\!\!\!\!\!\rightarrow X \land (\_ \diamond \_) = (\_ \fatsemi \_) \land e = \text{id}\, X \land inv = (\_^{\sim}) \end{array}$$

2. Symmetric set difference forms an abelian group over sets. The identity element is the empty set; the inverse function is the identity relation.

$$\begin{array}{l}[X] \vdash \\ \quad \exists\, \text{AbelianGroup}[\mathbb{P}\, X] \bullet \\ \qquad g = \mathbb{P}\, X \land (\_ \diamond \_) = (\_ \ominus \_) \land e = \varnothing \land inv = \text{id}(\mathbb{P}\, X) \end{array}$$

3. Arithmetic addition modulo n forms an abelian group over $0 \mathinner{..} n-1$. The identity element is zero; the inverse function is unary minus modulo n.

$$
\begin{array}{l}
n : \mathbb{N}_1 \vdash \\
\quad \exists\, \mathrm{AbelianGroup}[\mathbb{A}] \bullet \\
\qquad g = 0 \mathinner{..} n-1 \land (\_ \diamond \_) = (\_ + \_) \bmod n \\
\qquad \land\ e = 0 \land inv = (\ \lambda\, m : g \bullet \text{-}m \bmod n\ )
\end{array}
$$

4. Payments into a bank account should form an abelian group: there should be an inverse operation to *remove* money paid in.
5. Audit trails should *not* have an inverse operation, and hence should *not* form a group. Once an audit item has been added to the trail, it should not be possible to remove it.

Laws

■ **Law 23.3** A group has a unique inverse function.

$$
\begin{array}{l}
[X]\ \mathrm{Group}[X];\ inv_0 : X \pfun X \mid \\
\qquad g \lhd inv_0 \in g \to g \land (\ \forall\, x : g \bullet x \diamond inv_0\ x = e\ ) \vdash \\
\quad inv_0 = inv \\
[X]\ \mathrm{Group}[X];\ inv_0 : X \pfun X \mid \\
\qquad g \lhd inv_0 \in g \to g \land (\ \forall\, x : g \bullet inv_0\ x \diamond x = e\ ) \vdash \\
\quad inv_0 = inv
\end{array}
$$

■ **Law 23.4** Arithmetic multiplication modulo p, where p is prime, forms an abelian group over $1 \mathinner{..} p-1$.

$$
\begin{array}{l}
p : \mathrm{prime} \vdash \\
\quad \exists\, \mathrm{AbelianGroup}[\mathbb{A}] \bullet \\
\qquad g = 1 \mathinner{..} p-1 \land (\_ \diamond \_) = (\ \lambda\, n, m : g \bullet (n * m) \bmod p\ )
\end{array}
$$

---•

Distributing an abelian monoid

Intent

Distribute an Abelian monoid over a finite labelled set, to give the result of combining all the elements together using the function, to give a single element of the same type as the set elements.

Definition

$$
\begin{array}{l}
\mathrm{distributeOverLabelledSet}[L, X] == \\
\quad \{\ \mathrm{AbelianMonoid}[X];\ \diamond/ : (L \twoheadrightarrow X) \pfun X \mid \\
\qquad \diamond/ \in (L \twoheadrightarrow g) \to g \\
\qquad \land\ \diamond/\ \varnothing = e \\
\qquad \land\ (\ \forall\, l : L;\ x : g \bullet \diamond/\{l \mapsto x\} = x\ ) \\
\qquad \land\ (\ \forall\, f_1, f_2 : L \twoheadrightarrow g \mid \mathrm{dom}\, f_1 \cap \mathrm{dom}\, f_2 = \varnothing \bullet \\
\qquad\quad \diamond/(f_1 \cup f_2) = \diamond/f_1\ \diamond\ \diamond/f_2\ ) \bullet \\
\quad (L, g, (\_ \diamond \_)) \mapsto \diamond/\ \}
\end{array}
$$

The conventional name for the distributed form of a binary operator $\diamond$ is $\diamond/$.

We deal with the case of distributing a general (non-abelian) monoid, as *distributeOverSeq*, after we have defined sequences (§34).

Examples

1. Set union and intersection could be distributed over labelled sets

$$
\begin{array}{l}
[X]\ \vdash \mathrm{distributeOverLabelledSet}(\!|\ \\
\qquad g == \mathbb{P}\, X, \_ \diamond \_ == (\_ \cup \_)[X], e == \varnothing[X]\ |\!) \subseteq \cup/ \\
[X]\ \vdash \mathrm{distributeOverLabelledSet}(\!|\ \\
\qquad g == \mathbb{P}\, X, \_ \diamond \_ == (\_ \cap \_)[X], e == X\ |\!) \subseteq \cap/
\end{array}
$$

   However, since the operators are idempotent, their elements do not need to be labelled, so we use generalised union $\bigcup$ and generalised intersection $\bigcap$ instead. So, given a labelled set f, rather than using $\cup/f$ or $\cap/f$, we use $\bigcup(\mathrm{ran}\, f)$ and $\bigcap(\mathrm{ran}\, f)$ instead.
2. We distribute arithmetic sum and product over (finite) labelled sets below, to form $+/$ and $*/$. In §32 we introduce the functions Σ and Π for complete

distributed sum and product; they have been further generalised to extend to certain infinite cases.

Laws

Law 23.5 *distributeOverLabelledSet* is a total function from Abelian Monoids to total functions on finite labelled sets. (Proof of this law would involve the induction principle for finite sets, law 15.1.)

$$
\begin{array}{l}
[L, X] \ \vdash \text{distributeOverLabelledSet} \in \\
\qquad \{ \ \text{AbelianMonoid} \bullet (L, g, (\_ \diamond \_)) \ \} \to (L \twoheadrightarrow X) \to X
\end{array}
$$

Law 23.6 Distributing over the empty collection gives the identity element; distributing over the singleton collection gives the value; distributing over a collection of two gives the values combined by the binary operator.

$$
\begin{array}{l}
[L, X] \ \diamond / : (L \twoheadrightarrow X) \to X; \ \text{AbelianMonoid}[X] \mid \\
\qquad g = X \\
\qquad \wedge \diamond / = \text{distributeOverLabelledSet}(L, g, (\_ \diamond \_)) \vdash \\
\quad \diamond / \ \varnothing = e \\
[L, X] \ l : L; \ x : X; \ \diamond / : (L \twoheadrightarrow X) \to X; \ \text{AbelianMonoid}[X] \mid \\
\qquad g = X \\
\qquad \wedge \diamond / = \text{distributeOverLabelledSet}(L, g, (\_ \diamond \_)) \vdash \\
\quad \diamond / \ \{l \mapsto x\} = x \\
[L, X] \ k, l : L; \ x, y : X; \ \diamond / : (L \twoheadrightarrow X) \to X; \ \text{AbelianMonoid}[X] \mid \\
\qquad g = X \\
\qquad \wedge \#\{k, l\} = 2 \\
\qquad \wedge \diamond / = \text{distributeOverLabelledSet}(L, g, (\_ \diamond \_)) \vdash \\
\quad \diamond / \ \{k \mapsto x, l \mapsto y\} = x \diamond y \\
[L, X] \ j, k, l : L; \ x, y, z : X; \ \diamond / : (L \twoheadrightarrow X) \to X; \ \text{AbelianMonoid}[X] \mid \\
\qquad g = X \\
\qquad \wedge \#\{j, k, l\} = 3 \\
\qquad \wedge \diamond / = \text{distributeOverLabelledSet}(L, g, (\_ \diamond \_)) \vdash \\
\quad \diamond / \ \{j \mapsto x, k \mapsto y, l \mapsto z\} = x \diamond y \diamond z
\end{array}
$$

---•

Finite distributed sum

Intent

The operation of distributed sum applied to a finite labelled family of numbers finds the sum of those numbers.

Definition

$$+/[L] == \text{distributeOverLabelledSet}(L, \mathbb{R}, (\_ + \_))$$

The use of the function *distributeOverLabelledSet* in the definition covers all finite labelled sets. The application of this function may be pronounced *add up*.

Several Z authors have used a function with approximately this meaning, typically calling it $+/$ or Σ, but without formally defining it.

Examples

1. The empty distributed sum is zero; the distributed sum of a singleton is that element; the distributed sum of two items is the (binary) sum of those two.

$$\vdash +/\langle\,\rangle = 0$$
$$x : \mathbb{R} \vdash +/\langle x \rangle = x$$
$$x, y : \mathbb{R} \vdash +/\langle x, y \rangle = x + y$$

2. The sum of the first n integers, both in conventional mathematical notation, and in Z, is:

$$\sum_{m=1}^{n} m = n(n+1)/2$$

$$n : \mathbb{N} \vdash +/(\text{id}(1\,..\,n)) = n * (n+1) \div 2$$

3. The sum of the first n squares, both conventionally, and in Z, is:

$$\sum_{m=1}^{n} m^2 = n(n+1)(2n+1)/6$$

$$n : \mathbb{N} \vdash +/(\lambda\, m : 1\,..\,n \bullet m ** 2) = n * (n+1) * (2 * n + 1) \div 6$$

4. The sum of the first n cubes, both conventionally, and in Z, is:

$$\sum_{m=1}^{n} m^3 = n^2(n+1)^2/4$$

$$n : \mathbb{N} \vdash +/(\lambda\, m : 1\,..\,n \bullet m ** 3) = n ** 2 * (n+1) ** 2 \div 4$$

5. The sum of the first n powers, both conventionally, and in Z, is:

$$\sum_{m=0}^{n-1} x^m = \frac{1-x^n}{1-x}$$

$$n : \mathbb{N};\ x : \mathbb{R} \setminus \{1\} \vdash$$
$$+/(\ \lambda\, m : 0\,..\,(n-1) \bullet x ** m\ ) = (1 - x ** n) \div (1-x)$$

Finite distributed product

Intent

The operation of distributed product applied to a finite labelled family of numbers finds the product of those numbers.

Definition

$$*/[L] == \text{distributeOverLabelledSet}(L, \mathbb{R}, (\_ * \_))$$

The use of the function *distributeOverLabelledSet* in the definition covers all finite labelled sets. The application of this function may be pronounced *multiply up*.

A few Z authors have used a function with approximately this meaning, typically calling it $*/$ or Π, but without formally defining it.

Examples

1. The empty distributed product is unity; the distributed product of a singleton is that element; the distributed product of two items is the (binary) product of those two.

$$\vdash */\langle\ \rangle = 1$$
$$x : \mathbb{R} \vdash *\langle x \rangle = x$$
$$x, y : \mathbb{R} \vdash *\langle x, y \rangle = x * y$$

2. The product of the first n integers (the factorial function, §33) both conventionally, and in Z, is:

$$\prod_{m=1}^{n} m = n!$$

$$n : \mathbb{N} \vdash */(\mathrm{id}(1\,..\,n)) = \mathrm{factorial}\, n$$

———————————————•

Chapter 24

Homogeneous relations

In this chapter we explore properties of homogeneous relations, $X \leftrightarrow X$.

- vertices and roots
- well-rooted graphs: (wellRooted X)
- identity relation: id X
- reflexive relations and closure
- irreflexive relations and residue
- symmetric relations, closure and residue
- antisymmetric relations
- transitive relations
- transitive closure and reflexive transitive closure: $(\_^+)$, $(\_^*)$
- intransitive relations and residue
- locally finite relations: locallyFiniteOut X, locallyFiniteIn X, locallyFinite X
- equivalence relations: reflexive, symmetric, and transitive
- acyclic relations
- maximal iteration: do

A homogeneous relation has its domain and range of the same type; we first explore the meaning of the elementary set operations between these sets.

Some useful constraints on homogeneous relations take the form of assertions that particular patterns exist either always or never in the relation. The patterns we consider are:

- *reflexive* pairs, elements related to themselves: $x \mapsto x \in r$
- *symmetric* pairs, elements are either related or unrelated in both directions, so that for distinct x and y we have $x \mapsto y \in r \Leftrightarrow y \mapsto x \in r$

- *transitive* triples, so that $x \mapsto y \in r \wedge y \mapsto z \in r \Rightarrow x \mapsto z \in r$

These patterns give rise to six standard properties, of being *reflexive*, *irreflexive*, *symmetric*, *antisymmetric*, *transitive* or *intransitive*, according to whether they arise always or never.

We define the operations to make these patterns apply: the P *closure* of a relation r is the *smallest superset* of r that has property P; the P *residue* of a relation r is the *largest subset* of r that has property P.

In general, a relation may have some pairs that have the specified properties, and some that do not. When specifying, it is good practice always to check if a particular relation can be chosen so that *all* or *none* of its pairs have the relevant properties. Such a choice can help avoid problems of over-specification and proliferation of special cases. One can also immediately apply the relevant laws and thus raise the level of one's thinking.

Design choices

Abstraction

As the British philosopher and logician Bertrand Russell, 1872–1970, observed, pure mathematics may be defined as "the subject in which we never know what we are talking about" (in his essay Mathematics and the Metaphysicians, in [Russell 1929]). That is, we are dealing with *abstraction*, and by drawing the maximum of conclusions from the minimum of assumptions we maximise the usefulness of our thought processes. We can then apply our theories in a wide variety of different particular circumstances.

When thinking about our abstractions we need to have some sort of a mental model, but we must be careful not to let that mental model be too detailed, as it may then suggest unjustified conclusions. In developing our mental model it may be helpful to consider two or more contrasting particular cases, to guard against that danger.

Directed graph models

One mental model of homogeneous relations, and a possible application of the theory, is the *directed graph*, or *digraph*, where we have a set of *vertices* joined by directed *arcs*. The set X corresponds to the set of possible vertices, and the

elements of $X \leftrightarrow X$ correspond to the arcs between them. An *undirected graph* corresponds to a similar relation constrained to be *symmetric*.

An alternative mental model is given by a comparison algorithm. Here we take the set X as being the possible arguments to a two-argument function that returns the answer 'yes' or 'no'. The relation $X \leftrightarrow X$ then corresponds to the pairs of argument that give the result 'yes'. This mental model, though still mathematically imperfect, is preferable to that of the graph because it suggests less in the way of extraneous constraints. For example, with the idea of the graph in our minds we might suppose that each vertex inevitably has only a finite number of immediate neighbours. This need not be the case however; if we want that property we must specify it.

A theory of homogeneous relations can be developed specifically to support graph theory. Graph theory in general requires a richer structure than a homogeneous relation on its own provides. For example if it is desired to provide an explicit representation of isolated points, or to allow multiple edges between any particular pair of vertices, a single relation is not enough. Graph theory can also be applied to problems like that of flows in networks, where each arc or edge can be associated with some attribute, such as a numeric value.

If one's specification needs such richness, it is not difficult to declare the necessary structures, but a useful start to graph theory can be developed on the simpler basis of homogeneous relations. So we make a few elementary definitions and give some of their immediate consequences. There is a difference in emphasis, too, in that many of the concerns of graph theory are with *finite* graphs, whereas here we concentrate on the general case, giving definitions and results that apply equally whether the graph is finite or infinite.

Design choices in definitions

Z is a powerful and flexible notation, and therefore there is often no best way of defining things. We have a number of choices in this chapter, which we can exemplify by considering the possible ways of defining reflexivity.

1. If we have some set $a : \mathbb{P}\, X$, and we want the expression *reflexive* a to mean the set of all reflexive homogeneous relations using the set a, we can choose between at least the following definitions:
   - $\mathit{reflexive}\ a == \{\ r : a \leftrightarrow a \mid \mathrm{id}\, a \subseteq r\ \}$
   - $\mathit{reflexive}\ a == \{\ r : X \leftrightarrow X \mid \mathrm{id}\, a \subseteq r\ \}$
   - $\mathit{reflexive}\ a == \{\ r : a \leftrightarrow a \mid \mathrm{id}(\mathrm{dom}\, r \cup \mathrm{ran}\, r) \subseteq r\ \}$

2. Continuing the previous example, we could abandon the restriction to a single parameter, and write with some appropriate definition, *reflexive*(X, a), where $a : \mathbb{P}\, X$.
3. *reflexive* could be a predicate asserting that relation r is reflexive.
4. We could define a schema

$$Reflexive[X] == [\ r : X \leftrightarrow X;\ a, b : \mathbb{P}\, X \mid \mathrm{id}\, a \subseteq r \subseteq b \times b\ ]$$

Our criteria for making the choice are (in no particular order):

- simplicity of definition
- adequacy of power to make the significant mathematical points that we wish to make
- amenability to proof
- compatibility with standard mathematical convention
- compatibility with existing Z convention, for example [Hayes 1993]
- applicability in an analogous way to related cases (in this case to the definition of 'irreflexive' and so on)

All the theory we have developed for our particular choice should readily be adaptable to any alternative approach that the reader may prefer.

Functions on the domain and range

Intent

A homogeneous relation $r : X \leftrightarrow X$ has its domain and range of the same type; so there are five simple functions that we can apply between them

- $\mathrm{dom}\, r \cup \mathrm{ran}\, r$: all *vertices* (also called nodes)
- $\mathrm{dom}\, r \cap \mathrm{ran}\, r$: interior vertices; elements that are included in the relation in both senses: they are neither roots nor leaves.
- $\mathrm{dom}\, r \ominus \mathrm{ran}\, r$: exterior vertices; elements that are included in the relation in only one sense. In graph terms, these could be called the 'extrema' or the 'end nodes': they are either leaves or roots.
- $\mathrm{dom}\, r \setminus \mathrm{ran}\, r$: *leaves* (also called 'sources', or 'minima')
- $\mathrm{ran}\, r \setminus \mathrm{dom}\, r$: *roots* (also called 'sinks', 'or maxima')

These all have some meaning, as indicated. We formally define some but not all the corresponding functions, since it is straightforward for a specifier to do so for the rest whenever the need arises.

———————————————•

Vertices

Intent

The union of the domain and range corresponds to the whole set of elements that are anywhere in the relation. In graph terms, this is the set of vertices that are at either end of any arc.

Definition

vertices:

$$\mathrm{vertex}[X] == \lambda\, r : X \leftrightarrow X \bullet \mathrm{dom}\, r \cup \mathrm{ran}\, r$$

Laws

Law 24.1 All elements in the relation are vertices.

$$[X]\ x, y : X;\ r : X \leftrightarrow X \mid x \mapsto y \in r \vdash \{x, y\} \subseteq \mathrm{vertex}\, r$$

Law 24.2 *vertex* distributes through generalised union.

$$[X]\ \rho : \mathbb{P}(X \leftrightarrow X) \vdash \mathrm{vertex}(\bigcup \rho) = \bigcup(\mathrm{vertex}(\!|\ \rho\ |\!))$$
$$[X]\ r, s : X \leftrightarrow X \vdash \mathrm{vertex}(r \cup s) = \mathrm{vertex}\, r \cup \mathrm{vertex}\, s$$

Law 24.3 As a corollary of law 26.37 specialised to subsets, along with law 24.2, *vertex* is subset order-preserving.

$$[X]\ r, s : X \leftrightarrow X \mid r \subseteq s \vdash \mathrm{vertex}\, r \subseteq \mathrm{vertex}\, s$$

Law 24.4 Specialising the order-preserving law about bounds (law 26.36) to $f =$ *vertex*, gives

$$[X]\ \rho : \mathbb{P}(X \leftrightarrow X) \vdash \mathrm{vertex}(\bigcap \rho) \subseteq \bigcap(\mathrm{vertex}(\!|\ \rho\ |\!))$$
$$[X]\ r, s : X \leftrightarrow X \vdash \mathrm{vertex}(r \cap s) \subseteq \mathrm{vertex}\, r \cap \mathrm{vertex}\, s$$

Law 24.5 The vertices of a relation are the same as the vertices of its inverse.

$$[X]\ r : X \leftrightarrow X \vdash \mathrm{vertex}\, r = \mathrm{vertex}(r^{\sim})$$

———————————————•

Roots and leaves

Intent

Our definition has the arcs directed from the leaves towards the roots (just because it is technically more convenient to do it that way).

The names 'roots' (and 'leaves') are consistent with the image of the 'tree' and 'forest' special cases given below. We content ourselves with defining 'roots'. Several of the results about roots given below apply, dually, to leaves by inverting the relation.

Definition

roots:

The 'roots' of a graph are the vertices that an arc enters but does not leave.

$$\mathrm{root}[X] == \lambda\, r : X \leftrightarrow X \bullet \mathrm{ran}\, r \setminus \mathrm{dom}\, r$$

well-rooted graphs:

$$\mathrm{generic}\ (\mathrm{wellRooted}\ \_)$$
$$\mathrm{wellRooted}\, X == \{\ r : X \leftrightarrow X \mid \forall\, s_1 : \mathbb{P}_1\, r \bullet \mathrm{root}\, s_1 \neq \varnothing\ \}$$

A *well-rooted graph* is a graph for which any non-empty subgraph has a root.

Examples

1. $\{\ x, y : \mathbb{N} \mid x \leq y\ \} \in \mathrm{wellRooted}\, \mathbb{A}$

2. $\{ x, y : \mathbb{Z} \mid x \leq y \} \notin \text{wellRooted}\, \mathbb{A}$
3. $\{x \mapsto y, y \mapsto z, z \mapsto x\} \notin \text{wellRooted}\, X$

Identity relation

Intent

With the (generic) identity relation, every element is related to (only) itself. In graph terms, every element is connected to itself by an arc; there are no other arcs.

Definition

identity relation:

$$\text{generic}\,(\text{id}\,\_)$$
$$\text{id}\, X == \{ x : X \bullet x \mapsto x \}$$

Examples

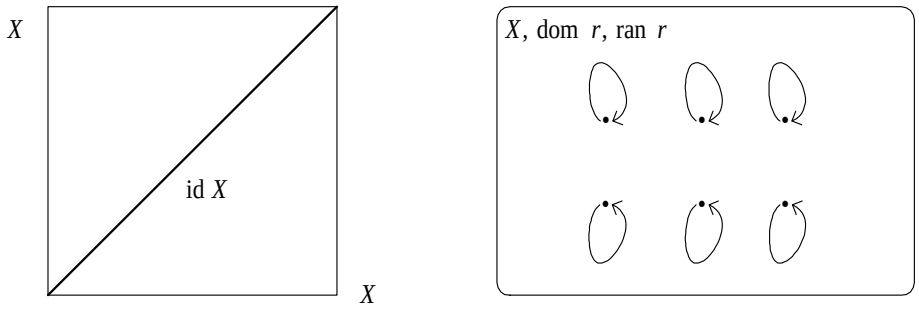


Figure 24.1 The identity relation.

1. $\text{id}\{x\} = \{x \mapsto x\}$
2. $\text{id}\{x, y\} = \{x \mapsto x, y \mapsto y\}$

Laws

Law 24.6 The identity relation is a bijection.

$$[X] \vdash \mathrm{id}\, X \in X \rightarrowtail\!\!\!\!\!\twoheadrightarrow X$$

Law 24.7 The domain of the identity relation is its parameter set. Dually, so is the range.

$$[X] \vdash \mathrm{dom}(\mathrm{id}\, X) = X = \mathrm{ran}(\mathrm{id}\, X)$$

Law 24.8 The identity relation is self-inverse.

$$[X] \vdash (\mathrm{id}\, X)^{\sim} = \mathrm{id}\, X$$

Law 24.9 Composing with the identity restricts the relation.

$$[X, Y]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow Y;\ b : \mathbb{P}\, Y \vdash \mathrm{id}\, a \fatsemi r \fatsemi \mathrm{id}\, b = a \lhd r \rhd b$$
$$[X]\ a, b : \mathbb{P}\, X \vdash \mathrm{id}\, a \fatsemi \mathrm{id}\, b = \mathrm{id}(a \cap b)$$

Law 24.10 The composition of a relation with its own inverse contains the identity.

$$[X, Y]\ r : X \leftrightarrow Y \vdash \mathrm{id}(\mathrm{dom}\, r) \subseteq r \fatsemi r^{\sim}$$
$$[X, Y]\ r : X \leftrightarrow Y \vdash \mathrm{id}(\mathrm{ran}\, r) \subseteq r^{\sim} \fatsemi r$$

Law 24.11 The identity relation is reflexive, symmetric, antisymmetric, and transitive; the identity relation is the only relation that is both symmetric and antisymmetric.

$$[X] \vdash \mathrm{id}\, X \in \mathrm{reflexive}\, X \cap \mathrm{symmetric}\, X \cap \mathrm{antisymmetric}\, X \cap \mathrm{transitive}\, X$$
$$[X] \vdash \mathrm{symmetric}\, X \cap \mathrm{antisymmetric}\, X = \mathbb{P}(\mathrm{id}\, X)$$

•

Reflexive relation

Intent

A reflexive relation is one where every element is related to (at least) itself. In graph terms, every element is connected to itself by an arc; there may be other arcs.

Definition

reflexive relations:

$$\text{generic}\,(\text{reflexive}\_)$$
$$\text{reflexive}\,X == \{\ r : X \leftrightarrow X \mid \text{id}\,X \subseteq r\ \}$$

Examples

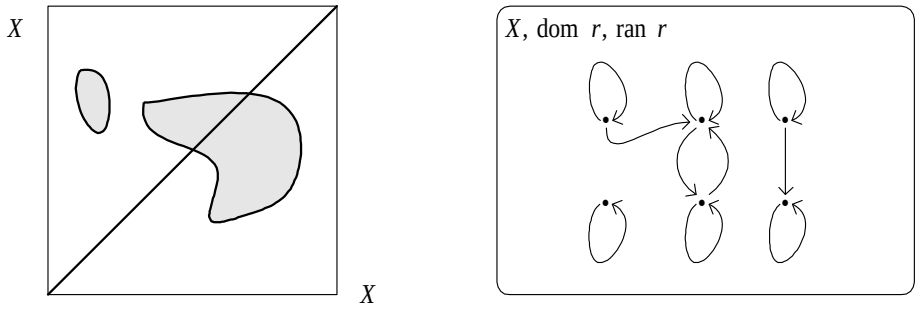


Figure 24.2 An example of a reflexive relation.

1. $\varnothing \in \text{reflexive}\,X$
2. $\{x \mapsto x\} \in \text{reflexive}\{x\}$
3. $\{x \mapsto x, x \mapsto y, y \mapsto y\} \in \text{reflexive}\{x, y\}$
4. $(\_ \subseteq \_) \in \text{reflexive}\,X$
5. A permutation of a sequence is one that has the same items, whether in the same order or not
   permutation : $\text{reflexive}(\text{seq}\,X)$
6. 'is parallel to' is reflexive

Laws

Law 24.12 The inverse of a reflexive relation is reflexive.

$$[X]\ r : \text{reflexive}\, X \vdash r^{\sim} \in \text{reflexive}\, X$$

Law 24.13 Any superset of a reflexive relation (that does not increase its domain) is reflexive. The union and intersection of reflexive relations is reflexive.

$$[X]\ s : X \leftrightarrow X;\ r : \text{reflexive}\, X \mid r \subseteq s \vdash s \in \text{reflexive}\, X$$
$$[X]\ \rho : \mathbb{P}(\text{reflexive}\, X) \vdash \bigcup \rho \in \text{reflexive}\, X$$
$$[X]\ \rho : \mathbb{P}_1(\text{reflexive}\, X) \vdash \bigcap \rho \in \text{reflexive}\, X$$

■ **Law 24.14** The composition of a relation with its inverse is reflexive on the relation's domain.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r \fatsemi r^{\sim} \in \text{reflexive}(\text{dom}\, r)$$
$$[X, Y]\ r : X \leftrightarrow Y \vdash r^{\sim} \fatsemi r \in \text{reflexive}(\text{ran}\, r)$$

•

Reflexive closure

Intent

The reflexive closure of a relation r is the smallest reflexive relation containing r.

Laws

Law 24.15 The reflexive closure of a relation is formed by unioning it with the identity.

$$[X]\ r : X \leftrightarrow X \vdash r \cup \text{id}\, X \in \text{reflexive}\, X$$

•

Irreflexive relation

Intent

An irreflexive relation is one with no element related to itself. In graph terms, no element is connected to itself by a single arc.

Definition

irreflexive relations:

generic (irreflexive _)

irreflexive $X == \{ r : X \leftrightarrow X \mid r \cap \mathrm{id}\, X = \varnothing \}$

Examples

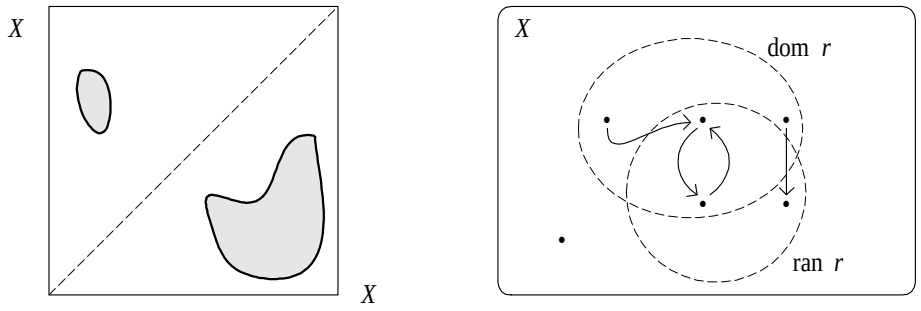


Figure 24.3 An example of an irreflexive relation.

1. $\varnothing \in \mathrm{irreflexive}\, X$
2. $x \neq y \Rightarrow \{x \mapsto y\} \in \mathrm{irreflexive}\, X$
3. $(\_ \neq \_) \in \mathrm{irreflexive}\, X$
4. $(\_ \subset \_) \in \mathrm{irreflexive}\, X$
5. $X^{2\times} \setminus \mathrm{id}\, X \in \mathrm{irreflexive}\, X$
6. 'is perpendicular to' is irreflexive
7. 'is sibling of' is irreflexive (§42)
8. A true anagram of a word has the same letters, but in a different order $trueAnagram : \mathrm{irreflexive}(\mathrm{seq}\, CHAR)$

Laws

Law 24.16 The inverse of an irreflexive relation is irreflexive.

$$[X]\ r : \text{irreflexive}\, X \vdash r^{\sim} \in \text{irreflexive}\, X$$

Law 24.17 Any subset of an irreflexive relation is irreflexive. The union of irreflexive relations is irreflexive.

$$[X]\ r : X \leftrightarrow X;\ s : \text{irreflexive}\, X \mid r \subseteq s \vdash r \in \text{irreflexive}\, X$$
$$[X]\ \rho : \mathbb{P}(\text{irreflexive}\, X) \vdash \bigcup \rho \in \text{irreflexive}\, X$$

■ **Law 24.18** A composition of two or more relations is irreflexive precisely when any cyclic permutation of the composition is irreflexive.

$$[X, Y]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow X \vdash r \fatsemi s \in \text{irreflexive}\, X \Leftrightarrow s \fatsemi r \in \text{irreflexive}\, Y$$
$$[X, Y, Z]\ r : X \leftrightarrow Y;\ s : Y \leftrightarrow Z;\ t : Z \leftrightarrow X \vdash$$
$$(r \fatsemi s \fatsemi t \in \text{irreflexive}\, X \Leftrightarrow s \fatsemi t \fatsemi r \in \text{irreflexive}\, Y)$$
$$\land\ (s \fatsemi t \fatsemi r \in \text{irreflexive}\, Y \Leftrightarrow t \fatsemi r \fatsemi s \in \text{irreflexive}\, Z)$$

Informally, this says that if, starting at any $x : X$ in the domain of r and travelling round the cycle of relations back to X, you cannot return to the element you started from (if there is no cyclic path from x), then you cannot find a cyclic path starting anywhere else along the cycle of relations.

•

Irreflexive residue

Intent

The irreflexive residue of a relation r is the largest irreflexive relation contained in r.

Laws

Law 24.19 The irreflexive residue of a relation is formed by subtracting the identity.

$$[X]\ r : X \leftrightarrow X \vdash r \setminus \mathrm{id}\, X \in \mathrm{irreflexive}\, X$$

Symmetric relation

Intent

A symmetric relation is the same in 'reverse', with its pairs swapped. Whereas a general homogeneous relation can be the model of a *directed* graph, a symmetric relation models an *undirected* graph, since all arcs go in both directions. These undirected arcs are called *edges*.

Definition

symmetric relations:

$$\mathrm{generic}\,(\mathrm{symmetric}\ \_)$$
$$\mathrm{symmetric}\, X == \{\ r : X \leftrightarrow X \mid r = r^{\sim}\ \}$$

Examples

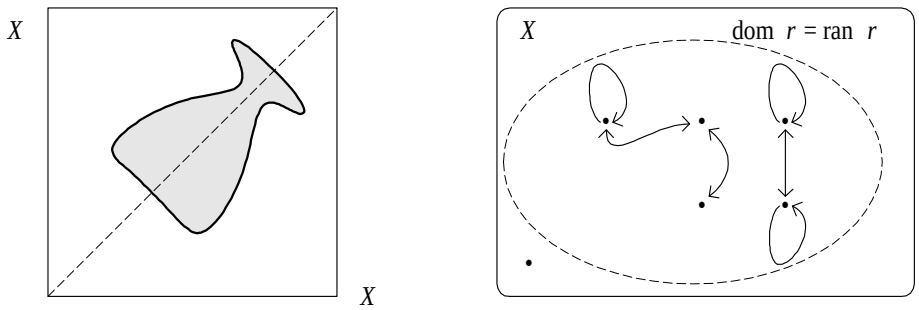


Figure 24.4 An example of a symmetric relation.

1. $\varnothing \in \mathrm{symmetric}\, X$
2. $\{x \mapsto x\} \in \mathrm{symmetric}\, X$
3. $\{x \mapsto y, y \mapsto x\} \in \mathrm{symmetric}\, X$
4. $(\_ \neq \_) \in \mathrm{symmetric}\, X$

5. 'is parallel to' is symmetric
6. 'is perpendicular to' is symmetric
7. 'is sibling of' is symmetric (§42)
8. *permutation* : symmetric(seq X)
9. *trueAnagram* : symmetric(seq *CHAR*)

Laws

Law 24.20 The union, intersection, and difference of symmetric relations is symmetric.

$$[X]\ \rho : \mathbb{P}(\text{symmetric}\ X) \vdash \bigcup \rho \in \text{symmetric}\ X$$
$$[X]\ \rho : \mathbb{P}_1(\text{symmetric}\ X) \vdash \bigcap \rho \in \text{symmetric}\ X$$
$$[X]\ r, s : \text{symmetric}\ X \vdash r \setminus s \in \text{symmetric}\ X$$

Law 24.21 A symmetric relation has no roots.

$$[X]\ r : \text{symmetric}\ X \vdash \text{root}\ r = \varnothing$$
$$[X]\ r : \text{symmetric}\ X \vdash \text{vertex}\ r = \text{dom}\ r = \text{ran}\ r$$

■ **Law 24.22** The composition of any relation with its inverse is symmetric.

$$[X, Y]\ r : X \leftrightarrow Y \vdash r \fatsemi r^\sim \in \text{symmetric}\ X$$
$$[X, Y]\ r : X \leftrightarrow Y \vdash r^\sim \fatsemi r \in \text{symmetric}\ Y$$

---•

Symmetric closure and residue

Intent

The symmetric closure of a relation r is the smallest symmetric relation containing r.

The symmetric residue of a relation r is the largest symmetric relation contained in r.

Definition

symmetric closure of a graph:

$$\text{symmetricClosure}[X] == \lambda\, r : X \leftrightarrow X \bullet r \cup r^{\sim}$$

For any homogeneous relation r the *symmetric closure* gives the underlying undirected graph.

Laws

Law 24.23 A symmetric relation is its own symmetric closure (its own underlying undirected graph).

$$[X]\ r : \text{symmetric}\, X \vdash r = \text{symmetricClosure}\, r$$

Law 24.24 The symmetric residue of a relation is formed by intersecting the relation with its inverse.

$$[X]\ r : X \leftrightarrow X \vdash r \cap r^{\sim} \in \text{symmetric}\, X$$

Antisymmetric relation

Intent

An antisymmetric relation is one that has no pairs that are both forward and 'reverse' between different elements; it may have reflexive pairs. In graph terms, no arc between distinct vertices goes in both directions; reflexive arcs are permitted, however.

Definition

antisymmetric relations:

$$\text{generic}\,(\text{antisymmetric}\,\_)$$
$$\text{antisymmetric}\, X == \{\ r : X \leftrightarrow X \mid r \cap r^{\sim} \subseteq \text{id}\, X\ \}$$

Examples

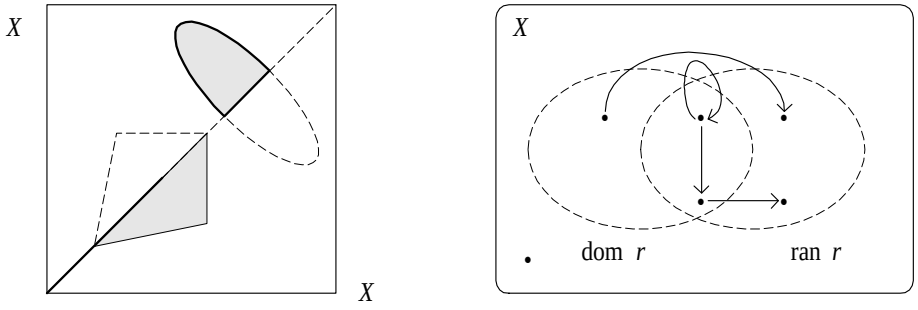


Figure 24.5 An example of an antisymmetric relation.

1. $\varnothing \in \text{antisymmetric}\, X$
2. $\{x \mapsto y\} \in \text{antisymmetric}\, X$
3. $(\_ \subseteq \_) \in \text{antisymmetric}\, X$
4. $(\_ \leq \_) \in \text{antisymmetric}\, \mathbb{A}$

Laws

Law 24.25 The inverse of an antisymmetric relation is antisymmetric.

$$[X]\ r : \text{antisymmetric}\, X \vdash r^{\sim} \in \text{antisymmetric}\, X$$

Law 24.26 Any subset of an antisymmetric relation is antisymmetric.

$$[X]\ r : X \leftrightarrow X;\ s : \text{antisymmetric}\, X \mid r \subseteq s \vdash r \in \text{antisymmetric}\, X$$

Transitive relation

Intent

A transitive relation is one where, for any pairs that could be composed together, the result of that composition is also in the relation. In graph terms, if there is a path of two arcs between three vertices, then there is a direct arc between the first and last vertices.

Definition

transitive relations:

$$\text{generic}\,(\text{transitive}\,\_)$$
$$\text{transitive}\,X == \{\ r : X \leftrightarrow X \mid r \fatsemi r \subseteq r\ \}$$

Examples

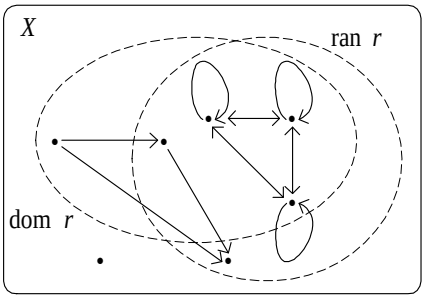


Figure 24.6 An example of a transitive relation.

1. $\varnothing \in \text{transitive}\,X$
2. $\{x \mapsto y, y \mapsto z, x \mapsto z\} \in \text{transitive}\,X$
3. $(\_ \subseteq \_) \in \text{transitive}\,X$
4. 'is parallel to' is transitive
5. 'is ancestor of' is transitive (§42)
6. *permutation* : transitive(seq X)

Laws

Law 24.27 An equivalent definition of *transitive* is:

$$[X] \vdash \text{transitive}\, X = \{\, r : X \leftrightarrow X \mid \forall\, x, y, z : X \mid x \mapsto y \in r \land y \mapsto z \in r \bullet x \mapsto z \in r \,\}$$

Law 24.28 The inverse of a transitive relation is transitive.

$$[X]\ r : \text{transitive}\, X \vdash r^{\sim} \in \text{transitive}\, X$$

■ **Law 24.29** Subsetting does not necessarily preserve transitivity, but the following do. Any restriction of a transitive relation is transitive; the intersection of two transitive relations is transitive.

$$[X]\ a, b : \mathbb{P}\, X;\ r : \text{transitive}\, X \vdash a \lhd r \rhd b \in \text{transitive}\, X$$
$$[X]\ r, s : \text{transitive}\, X \vdash r \cap s \in \text{transitive}\, X$$

■ **Law 24.30** Any relation that is both transitive and symmetric is also reflexive on its domain.

$$[X]\ r : \text{transitive}\, X \cap \text{symmetric}\, X \vdash r \in \text{reflexive}\ (\text{dom}\, r)$$

In graph terms, if two consecutive arcs can always be composed into one (transitive) and all arcs to immediate neighbours are two-way edges (symmetric) then there must be a single arc back to each domain element (antisymmetric).

Law 24.31 As a corollary of the previous law, there are no non-trivial relations that are transitive, irreflexive and symmetric.

$$[X] \vdash \text{transitive}\, X \cap \text{irreflexive}\, X \cap \text{symmetric}\, X = \{\varnothing\}$$

■ **Law 24.32** Any transitive function is the identity on its range. So the first application of the function does all the 'work', and subsequent applications have no further effect: the function is *idempotent*.

$$[X]\ f : (X \nrightarrow X) \cap \text{transitive}\, X \vdash \text{ran}\, f \lhd f = \text{id}(\text{ran}\, f)$$

■ **Law 24.33** Being a function is, in some sense, the 'opposite' of being transitive. An irreflexive, transitive function has no interior vertices; it is all roots and leaves.

$$[X]\ f : (X \pfun X) \cap \text{transitive}\, X \cap \text{irreflexive}\, X \vdash \text{dom}\, f \cap \text{ran}\, f = \varnothing$$

———————————————•

Transitive closures

Intent

The transitive closure of a relation r is the smallest transitive relation containing r.

The reflexive transitive closure of r is the smallest reflexive and transitive relation containing r. We get the reflexive transitive closure by unioning with the identity relation.

Definition

transitive closure:

$$\text{function}\ (\_^+)$$
$$\_^+\ [X] == \lambda\, r : X \rel X \bullet \bigcap \{\ t : \text{transitive}\, X \mid r \subseteq t\ \}$$

The comprehension in the definition of r^+ defines the set of transitive relations that have r as a subset. The use of the generalised intersection then gives us the smallest such possible set.

reflexive transitive closure:

$$\text{function}\ (\_^*)$$
$$\_^*\ [X] == \lambda\, r : X \rel X \bullet r^+ \cup \text{id}\, X$$

If we write r^*, we get a domain for the closed relation taken from the whole type. This may well be what we want. If not, the closure expression needs to be supplied with the relevant parameter, as $(\_^*)[a]r$. In such cases it is probably clearer to write $r^+ \cup \text{id}\, a$.

Examples

- $\varnothing^+ = \varnothing$
- $\varnothing^* = \mathrm{id}\, X$
- $\{x \mapsto y, y \mapsto z\}^+ = \{x \mapsto y, y \mapsto z, x \mapsto z\}$
- $(\_ \subseteq \_)^+ = (\_ \subseteq \_)^* = (\_ \subseteq \_)$
- $(\_ \subset \_)^+ = (\_ \subset \_)$
- $(\_ \subset \_)^* = (\_ \subseteq \_)$

Laws

Law 24.34 The closures are total functions. Their ranges are the sets of all appropriately transitive relations.

$$[X] \ \vdash (\_^+)[X] \in (X \leftrightarrow X) \twoheadrightarrow \mathrm{transitive}\, X$$
$$[X] \ \vdash (\_^*)[X] \in (X \leftrightarrow X) \twoheadrightarrow (\mathrm{reflexive}\, X \cap \mathrm{transitive}\, X)$$

Law 24.35 The closures are idempotent.

$$[X]\ r : X \leftrightarrow X \vdash r^{++} = r^+$$
$$[X]\ r : X \leftrightarrow X \vdash r^{+*} = r^*$$
$$[X]\ r : X \leftrightarrow X \vdash r^{**} = r^*$$
$$[X]\ r : X \leftrightarrow X \vdash r^{*+} = r^*$$

Law 24.36 The domain and range of a relation are unchanged under transitive closure.

$$[X]\ r : X \leftrightarrow X \vdash \mathrm{dom}(r^+) = \mathrm{dom}\, r$$
$$[X]\ r : X \leftrightarrow X \vdash \mathrm{ran}(r^+) = \mathrm{ran}\, r$$

Law 24.37 The transitive closure of a relation is the union of all its non-trivial iterations. The reflexive transitive closure of a relation is the union of all its iterations, including the trivial zero iteration (or identity).

$$[X]\ r : X \leftrightarrow X \vdash r^+ = \bigcup \{\ n : \mathbb{N}_1 \bullet r^n\ \}$$
$$[X]\ r : X \leftrightarrow X \vdash r^* = \bigcup \{\ n : \mathbb{N} \bullet r^n\ \}$$

■ **Law 24.38** Transitive closure is subset order-preserving (§26).

$$[X]\ r, s : X \leftrightarrow X \mid r \subseteq s \vdash r^+ \subseteq s^+$$
$$[X]\ r, s : X \leftrightarrow X \mid r \subseteq s \vdash r^* \subseteq s^*$$

Law 24.39 Specialising the order-preserving law about bounds (law 26.36) to $f = \_^+$, and to $f = \_^*$, gives

$$[X]\ \rho : \mathbb{P}(X \leftrightarrow X) \vdash \bigcup\{\ r : \rho \bullet r^+\ \} \subseteq (\bigcup \rho)^+$$
$$[X]\ r, s : X \leftrightarrow X \vdash r^+ \cup s^+ \subseteq (r \cup s)^+$$

$$[X]\ \rho : \mathbb{P}(X \leftrightarrow X) \vdash \bigcup\{\ r : \rho \bullet r^*\ \} \subseteq (\bigcup \rho)^*$$
$$[X]\ r, s : X \leftrightarrow X \vdash r^* \cup s^* \subseteq (r \cup s)^*$$

$$[X]\ \rho : \mathbb{P}(X \leftrightarrow X) \vdash (\bigcap \rho)^+ \subseteq \bigcap\{\ r : \rho \bullet r^+\ \}$$
$$[X]\ r, s : X \leftrightarrow X \vdash (r \cap s)^+ \subseteq r^+ \cap s^+$$

$$[X]\ \rho : \mathbb{P}(X \leftrightarrow X) \vdash (\bigcap \rho)^* \subseteq \bigcap\{\ r : \rho \bullet r^*\ \}$$
$$[X]\ r, s : X \leftrightarrow X \vdash (r \cap s)^* \subseteq r^* \cap s^*$$

■ **Law 24.40** The transitive closure of a relation is irreflexive precisely when all the iterates of that relation are irreflexive.

$$[X]\ r : X \leftrightarrow X \vdash r^+ \in \text{irreflexive}\, X \Leftrightarrow (\ \forall\, n : \mathbb{N}_1 \bullet r^n \in \text{irreflexive}\, X\ )$$

———————————————•

Intransitive relation

Intent

An intransitive relation is one where no pair can be obtained by linking together other pairs end to end. In graph terms, if there is a direct arc between two vertices, then there is no path of two or more arcs between them.

Definition

intransitive relations:

$$\text{generic}\,(\text{intransitive}\,\_)$$
$$\text{intransitive}\,X == \{\ r : X \leftrightarrow X \mid r \cap (r \fatsemi r^+) = \varnothing\ \}$$

Examples

1. $\varnothing \in \text{intransitive}\,X$
2. $\{x \mapsto y, y \mapsto z\} \in \text{intransitive}\,X$
3. $\lambda\, i : \mathbb{Z} \bullet i + 1 \in \text{intransitive}\,\mathbb{Z}$
4. The *mother* relation is intransitive (§42).

Laws

Law 24.41 An intransitive relation has no transitive triples.

$$[X]\ r : \text{intransitive}\,X;\ x, y, z : X \mid \{x \mapsto y, y \mapsto z\} \subseteq r \vdash x \mapsto z \notin r$$

Law 24.42 The inverse of an intransitive relation is intransitive.

$$[X]\ r : \text{intransitive}\,X \vdash r^\sim \in \text{intransitive}\,X$$

Law 24.43 Any subset of an intransitive relation is intransitive.

$$[X]\ r : X \leftrightarrow X;\ s : \text{intransitive}\,X \mid r \subseteq s \vdash r \in \text{intransitive}\,X$$

Law 24.44 Intransitive relations are acyclic.

$$[X]\ \vdash \text{intransitive}\,X \subseteq \text{acyclic}\,X$$

---•

Intransitive residue

Intent

The *intransitive residue* of a relation is the intransitive relation that remains when all the transitive parts are removed. In graph terms, the intransitive residue of a graph has any arc directly connecting two vertices removed if there is also an indirect, multi-arc path between them. (Note that sometimes the removed direct arc may have been part of the indirect path.)

Definition

intransitive residue:

$$\text{intransitiveResidue}[X] == \lambda\, r : X \leftrightarrow X \bullet r \setminus (r \fatsemi r^+)$$

Examples

1. $\text{intransitiveResidue}((\_ \leq \_) \cap (\mathbb{Z} \times \mathbb{Z})) = \varnothing$
2. $\text{intransitiveResidue}((\_ < \_) \cap (\mathbb{R} \times \mathbb{R})) = \varnothing$

Laws

Law 24.45 An intransitive residue is an intransitive relation.

$$[X]\ r : X \leftrightarrow X \vdash \text{intransitiveResidue}\, r \in \text{intransitive}\, X$$

Law 24.46 Reflexive relations and symmetric relations have no intransitive residue.

$$[X]\ r : \text{reflexive} \vdash \text{intransitiveResidue}\, r = \varnothing$$
$$[X]\ r : \text{symmetric} \vdash \text{intransitiveResidue}\, r = \varnothing$$

Law 24.47 A relation is acyclic precisely when it is contained within the transitive closure of its intransitive residue.

$$[X]\ r : X \leftrightarrow X \vdash r \in \text{acyclic}\, X \Leftrightarrow r \subseteq (\text{intransitiveResidue}\, r)^+$$

Law 24.48 A relation is transitive precisely when it contains the transitive closure of its intransitive residue.

$$[X]\ r : X \leftrightarrow X \vdash r \in \text{transitive}\, X \Leftrightarrow (\text{intransitiveResidue}\, r)^+ \subseteq r$$

Law 24.49 Hence a relation is acyclic and transitive precisely when it equals transitive closure of its intransitive residue. (It is also an irreflexive partial order, §26.)

$$[X]\ r : X \leftrightarrow X \vdash r \in \text{acyclic}\, X \cap \text{transitive}\, X \Leftrightarrow r = (\text{intransitiveResidue}\, r)^+$$

Vertex finiteness

Intent

We define the case where there is only a finite number of arcs entering or leaving a vertex. If all vertices have finite degree we call the relation *locally finite*.

Definition

generic 80(locallyFiniteOut _)
$\text{locallyFiniteOut}\, X == \{\ r : X \leftrightarrow X \mid \forall x : X \bullet \text{finite}(\text{successors}\, r\ x)\ \}$

generic 80(locallyFiniteIn _)
$\text{locallyFiniteIn}\, X == \{\ r : X \leftrightarrow X \mid r^\sim \in \text{locallyFiniteOut}\, X\ \}$

generic 80(locallyFinite _)
$\text{locallyFinite}\, X == \{\ r : X \leftrightarrow X \mid \text{symmetricClosure}\, r \in \text{locallyFiniteOut}\, X\ \}$

Equivalence relation

Intent

An equivalence relation is a relation that is reflexive, symmetric, and transitive.

Definition

equivalence relations:

generic (equivalence _)
equivalence X == reflexive $X \cap$ symmetric $X \cap$ transitive X

Examples

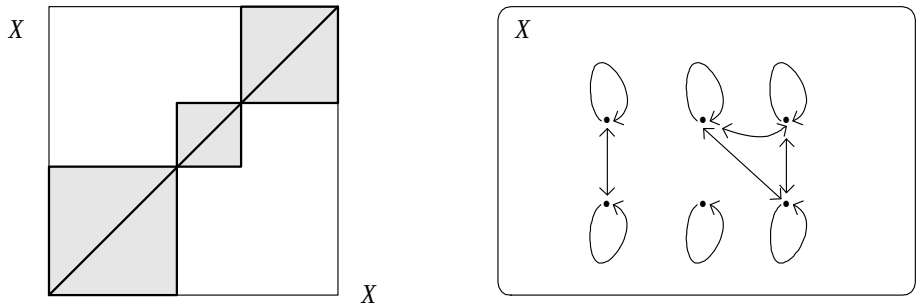


Figure 24.7 An example of an equivalence relation.

1. id $X \in$ equivalence X: 'equality' is the prototypical equivalence relation. Each element is the sole member of its own equivalence class.
2. $X^{2\times} \in$ equivalence X: there is a single equivalence class consisting of all elements
3. 'generates the same component as' is an equivalence relation on vertices
4. 'exists a bijection between' is an equivalence relation on sets. For finite sets, each equivalence class (see law 24.51) contains sets all having the same cardinality; the existence of this equivalence relation is a way to extend the definition of cardinality to infinite sets.
5. *permutation* $\in$ equivalence(seq X)

6. if we interpret an ordered pair of naturals as a rational number, 'is the same rational as' $\in$ equivalence($\mathbb{N}_1{}^{2\times}$)
7. 'is parallel to' is an equivalence relation

Laws

Law 24.50 The symmetric residue of a preorder is an equivalence relation.

$$[X]\ r : \text{preorder}\ X \vdash r \cap r^{\sim} \in \text{equivalence}\ X$$

Law 24.51 An equivalence relation partitions its generic set into equivalence classes.

$$[L, X]\ r : \text{equivalence}\ X \vdash \\ \quad \exists f : L \twoheadrightarrow \mathbb{P}\ X \bullet f\ \text{partition}\ X \land r = \bigcup \{\ a : \text{ran}\ f \bullet a^{2\times}\ \}$$

Law 24.52 Any partition can be used to generate an equivalence relation.

$$[L, X]\ f : L \twoheadrightarrow \mathbb{P}\ X \mid f\ \text{partition}\ X \vdash \bigcup \{\ a : \text{ran}\ f \bullet a^{2\times}\ \} \in \text{equivalence}\ X$$

---•

Acyclic relation

Intent

An *acyclic* relation is one where no vertex is related to itself by one or more applications of the relation.

Definition

acyclic relations:

$$\text{generic}\ (\text{acyclic}\ \_)$$
$$\text{acyclic}\ X == \{\ r : X \leftrightarrow X \mid r^{+} \in \text{irreflexive}\ X\ \}$$

The transitive closure of an acyclic relation is irreflexive.

Examples

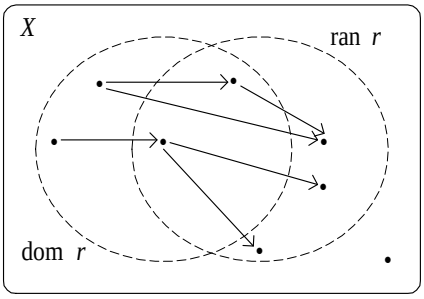


Figure 24.8 An example of an acyclic relation.

1. $\varnothing \in \text{acyclic}\, X$
2. $(\_ \subset \_) \in \text{acyclic}\, X$
3. The *parent* relation is acyclic (§42).
4. The inheritance relation in an object oriented language such as Eiffel (with multiple inheritance) forms an acyclic graph of classes.

Laws

Law 24.53 The inverse of an acyclic relation is acyclic.

$$[X]\ r : \text{acyclic}\, X \vdash r^{\sim} \in \text{acyclic}\, X$$

Law 24.54 A subset of an acyclic relation is acyclic.

$$[X]\ r : X \leftrightarrow X;\ s : \text{acyclic}\, X \mid r \subseteq s \vdash r \in \text{acyclic}\, X$$

Law 24.55 All finite acyclic relations are well-rooted; all well-rooted relations are acyclic.

$$[X]\ \vdash \text{acyclic}\, X \cap (X \nrightarrow X) \subseteq \text{wellRooted}\, X \subseteq \text{acyclic}\, X$$

Law 24.56 Any transitive, irreflexive relation is acyclic; any acyclic relation is irreflexive and antisymmetric.

$$[X] \vdash \text{transitive}\, X \cap \text{irreflexive}\, X \subseteq \text{acyclic}\, X \subseteq \text{irreflexive}\, X \cap \text{antisymmetric}\, X$$

In graph terms, if two consecutive arcs can always be composed into one (transitive) and there are no reflexive arcs (irreflexive) then there can be no two-way paths (acyclic).

Law 24.57 The transitive closure of an acyclic relation is an irreflexive partial order; the reflexive transitive closure of an acyclic relation is a reflexive partial order (§26).

$$[X] \vdash (\_^+)(\!|\, \text{acyclic}\, X \,|\!) = \text{irreflexiveOrder}\, X$$
$$[X] \vdash (\_^*)(\!|\, \text{acyclic}\, X \,|\!) = \text{reflexiveOrder}\, X$$

Maximal iteration

Intent

r^* contains all pairs of values which may be obtained by composing together elements of r, and *do* r contains all such pairs where the process of composition has gone as far as possible, in the sense that the range value obtained is no longer in the domain of r.

Although the function *do* is defined for any relation $r : X \leftrightarrow X$, the most useful applications are those where r is a non-total function. In that case, *do* r corresponds to the action of iterating the function as often as possible for each value in its domain. The iteration of the function 'halts' for the values in $\text{dom}(\text{do}\, r)$ and 'halts everywhere' if $\text{dom}(\text{do}\, r) = X$

The significance of do has been well developed by such theorists as the Dutch computer scientist Edsgar Dijkstra, 1930–2002 [Dijkstra 1976]. His treatment uses "non-determinacy", which corresponds with the use of relations as functions, where the application of a "non-deterministic function" to its argument corresponds to a choice from the image of the arguments through the relation. He also uses "guards", which correspond to domain restrictions. The separate clauses of his "do" and "if" statements are put together in a way that corresponds to set union. With this understanding, our do and Dijkstra's "do" are identical

Definition

$$\mathrm{do}[X] == \lambda\, r : X \leftrightarrow X \bullet r^* \rhd \mathrm{dom}\, r$$

Examples

1. The factorial function (also defined later):

$$n : \mathbb{N} \vdash \mathrm{do}(\ \lambda\, m : \mathbb{N}_1;\ k : \mathbb{Z} \bullet (m - 1, k * m)\ )(n, 1) = (0, \mathrm{factorial}\, n)$$

2. The Greek/Egyptian mathematician Euclid ($\sim$ 325B.C.–unknown) gave an algorithm to calculate the greatest common divisor. Euclid wrote in the Greek language, but his home was in Egypt, where many people spoke Greek at that time. His ancestry is unknown, so could equally well have been Egyptian or Greek. More importantly, he was heir not only to the earlier Greek mathematical tradition, but also to the much longer ancient Egyptian mathematical tradition, to which the Greeks themselves frequently paid tribute. Euclid's algorithm can be written as:

$$\begin{array}{l} p, q : \mathbb{N}_1 \vdash \\ \quad \mathrm{do}((\ \lambda\, n, m : \mathbb{N}_1 \mid m < n \bullet (n - m, m)\ ) \\ \qquad\qquad \cup (\ \lambda\, n, m : \mathbb{N}_1 \mid n < m \bullet (n, m - n))\ )(p, q) \\ \qquad = (\gcd(p, q), \gcd(p, q)) \end{array}$$

3. In computer arithmetic on binary integers, multiplication can be carried out by left-shifting (doubling), right-shifting (halving), testing the lost bit on right-shifting (modulo 2) and adding. So using the operations of $n*2$, $n \,\mathrm{div}\, 2$, and $n \bmod 2$ respectively, we can represent the process of multiplication of integer x by y with the equation

$$\begin{array}{l} (x * y, y', 0) = \\ \quad \mathrm{do}(\ \lambda\, a, b : \mathbb{N};\ c : \mathbb{N}_1 \bullet \\ \qquad\qquad (a + \mathrm{if}\ c \bmod 2 = 1\ \mathrm{then}\ b\ \mathrm{else}\ 0, b * 2, c \,\mathrm{div}\, 2)\ ) \\ \qquad (0, x, y) \end{array}$$

where y' can be discarded.

Laws

Law 24.58 *do* is a total function.

$$[X] \ \vdash \mathrm{do}[X] \in (X \leftrightarrow X) \to (X \leftrightarrow X)$$

Law 24.59 *do* r is the smallest relation that contains $\mathrm{id}(X \setminus \mathrm{dom}\, r)$ and that is unaltered by having r itself relationally composed with it.

$$[X] \ r : X \leftrightarrow X \vdash \mathrm{do}\, r = \bigcap \{ \ s : X \leftrightarrow X \mid \mathrm{id}(X \setminus \mathrm{dom}\, r) \subseteq s \wedge r \fatsemi s \subseteq s \ \}$$

Law 24.60 Maximally iterating the empty relation gives the identity relation.

$$[X] \ \vdash \mathrm{do}[X] \varnothing = \mathrm{id}\, X$$

Law 24.61 Maximally iterating a function yields a function, and is partitioned by the iterates (§31).

$$[X] \ f : X \nrightarrow X \vdash \mathrm{do}\, f \in X \nrightarrow X$$
$$[X] \ f : X \nrightarrow X \vdash (\ \lambda\, n : \mathbb{N} \bullet f^n \rhd \mathrm{dom}\, f \ )\, \mathrm{partition}(\mathrm{do}\, f)$$

Law 24.62 Alternative definitions of *do* in terms of relational iteration (§31)

$$[X] \ r : X \leftrightarrow X; \ n : \mathbb{N} \vdash$$
$$\mathrm{do}\, r = \bigcup \{ \ m : \mathbb{N} \mid m < n \bullet r^m \rhd \mathrm{dom}\, r \ \} \cup (r^n \fatsemi \mathrm{do}\, r)$$
$$[X] \ r : X \leftrightarrow X \vdash \mathrm{do}\, r = \bigcup \{ \ n : \mathbb{N} \bullet r^n \rhd \mathrm{dom}\, r \ \}$$

•

Chapter 25

Connected graphs, forests and trees

In this chapter we further develop the theory of homogeneous binary relations in a direction useful for graph theory.

- strongly connected graphs, connected graphs
- non-empty connected graphs
- components
- forests
- trees

Connected graph

Intent

A relation is *strongly connected* if every vertex is connected to every other vertex by multiple applications of the relation. Hence the transitive closure of the relation is completely full on the vertex set.

A relation is *connected* if every vertex is connected to every other vertex by multiple applications of the relation and its inverse. Hence a relation is connected if its underlying symmetric graph is strongly connected.

The internally connected but mutually disconnected parts of a graph are its *component subgraphs*.

Definition

strongly connected relations:

$$\text{generic}\,(\text{stronglyConnected}\,\_)$$
$$\text{stronglyConnected}\,X == \{\ r : X \leftrightarrow X \mid r^+ = (\text{vertex}\,r)^{2\times}\ \}$$

connected relations:

$$\text{generic}\,(\text{connected}\,\_)$$
$$\text{connected}\,X == \{\ r : X \leftrightarrow X \mid \text{symmetricClosure}\,r \in \text{stronglyConnected}\,X\ \}$$
$$\text{connected}_1\,X == \text{connected}\,X \setminus \varnothing$$

component:

We define the component subgraph of any graph that is generated by one of its vertices:

$$\text{component}[X] == \lambda\, r : X \leftrightarrow X \bullet \lambda\, x : X \bullet$$
$$\{\ p : r \mid x \mapsto p.1 \in (\text{symmetricClosure}\,r)^+\ \}$$

Examples

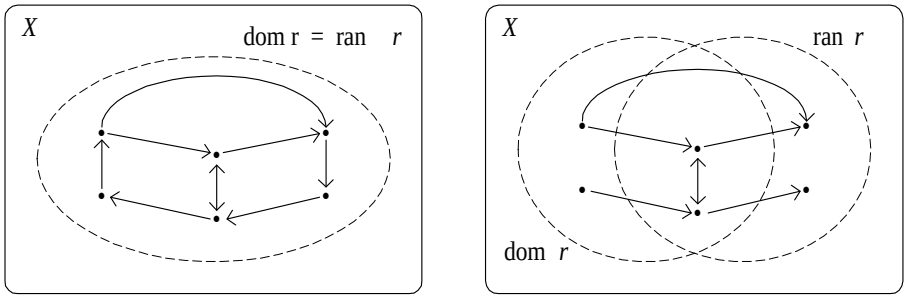


Figure 25.1 an example of (a) a strongly connected relation (b) a connected relation.

Laws

Law 25.1 A strongly connected relation has no roots.

$$[X]\ r : \text{stronglyConnected}\,X \vdash \text{root}\,r = \varnothing$$

Law 25.2 The only strongly connected acyclic relation is the empty one.

$$[X]\ r : \mathrm{stronglyConnected}\, X \cap \mathrm{acyclic}\, X \vdash r = \varnothing$$

This result shows that being acyclic and being strongly connected can be regarded as opposite extremes; in general a graph might be neither.

Law 25.3 The union of two non-empty graphs is not connected precisely when their vertex sets are disjoint.

$$[X]\ r, s : \mathrm{connected}_1\, X \vdash r \cup s \notin \mathrm{connected}\, X \Leftrightarrow \mathrm{disjoint}\langle \mathrm{vertex}\, r, \mathrm{vertex}\, s\rangle$$

Law 25.4 If an element is not a vertex of a graph, it generates the empty component.

$$[X]\ r : X \leftrightarrow X;\ x : X \mid x \notin \mathrm{vertex}\, r \vdash \mathrm{component}\, r\ x = \varnothing$$

Law 25.5 In general there may be any number of components, which together make up the original graph. The roots of the graph are divided up amongst the components:

$$[X]\ r : X \leftrightarrow X \vdash r = \bigcup\{\ x : \mathrm{vertex}\, r \bullet \mathrm{component}\, r\ x\ \}$$
$$[X]\ r : X \leftrightarrow X \vdash \mathrm{root}\, r = \bigcup\{\ x : \mathrm{vertex}\, r \bullet \mathrm{root}(\mathrm{component}\, r\ x)\ \}$$

Law 25.6 A connected graph has only one component.

$$[X]\ r : \mathrm{connected}\, X;\ x : X \mid x \in \mathrm{vertex}\, r \vdash \mathrm{component}\, r\ x = r$$

Forest

Intent

An acyclic function has each node connected to at most one other node, its *parent*. This is called a *forest*.

Definition

forests:

$$\text{generic}\,(\text{forest}\,\_)$$
$$\text{forest}\,X == \text{acyclic}\,X \cap (X \nrightarrow X)$$

Examples

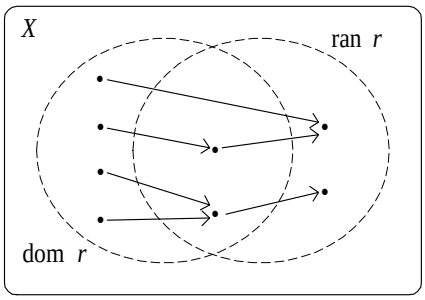


Figure 25.2 An example of a forest.

1. Figure 25.2 shows an example of a forest.
2. A directory structure in a collection of disjoint hierarchical file systems forms a forest.
3. The *mother* and *father* relations are forests (§42).

Laws

Law 25.7 Forests are intransitive.

$$[X] \vdash \text{forest}\,X \subseteq \text{intransitive}\,X$$

Law 25.8 Any subset of a forest is a forest.

$$[X]\ r : X \leftrightarrow X;\ s : \text{forest}\,X \mid r \subseteq s \vdash r \in \text{forest}\,X$$

Law 25.9 The union of two forests with disjoint ranges is acyclic.

$$[X]\ r, s : \text{forest}\,X \mid \text{disjoint}\langle \text{ran}\,r, \text{ran}\,s\rangle \vdash r \cup s \in \text{acyclic}\,X$$

Tree

Intent

A connected forest is a *tree*.

Definition

trees:

$$\text{generic}\,(\text{tree}\,\_)$$
$$\text{tree}\,X == \text{forest}\,X \cap \text{connected}\,X$$

Examples

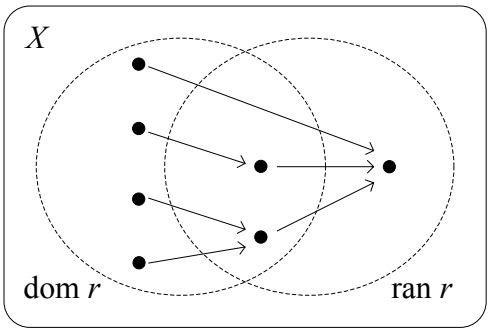


Figure 25.3 An example of a tree.

1. Figure 25.3 shows an example of a tree.
2. A simple directory structure in a hierarchical file system forms a tree.
3. The inheritance relation in an object oriented language such as Smalltalk (with single inheritance, and a single root class) forms a tree of classes.

Laws

Law 25.10 The components of a forest are trees.

$$[X]\ r : \text{forest}\,X;\ x : X \vdash \text{component}\,r\ x \in \text{tree}\ X$$

Law 25.11 A tree has at most one root.

$$[X]\ r : \operatorname{tree} X;\ x, y : X \mid \{x, y\} \subseteq \operatorname{root} r \vdash x = y$$

Law 25.12 Any finite set of vertices in a tree has a least upper bound in the reflexive transitive closure of the tree.

$$[X]\ r : \operatorname{tree} X;\ a : \mathbb{F}\, X \mid a \in \operatorname{dom} r \vdash \exists\, b : \operatorname{ran} r \bullet a \mapsto b \in \operatorname{lub} r^*$$

---•

Chapter 26

Orders

The subset relation $\subseteq$ has the property that if $a \subseteq b$, and $b \subseteq c$, then $a \subseteq c$; it is *transitive*. It also has the property that the only way for both $a \subseteq b$ and $b \subseteq a$, is to have $a = b$; it is *antisymmetric*. These are the properties of a well-known mathematical construct: a *(partial) order*. (In Z parlance the word 'partial' is redundant; it should be used only where it is necessary to draw attention to the *absence* of the word 'total'. 'Partial' does *not* mean 'non-total'.)

This chapter is devoted to properties of orders.

- orders, reflexive orders, irreflexive orders
- posets, non-empty posets
- total orders, reflexive total orders, irreflexive total orders
- chains, non-empty chains
- preorders
- minimum, maximum
- greatest lower bound, least upper bound: glb, lub
- well orders, reflexive well orders, irreflexive well orders
- well founded chains, non-empty well founded chains
- graph preserving maps, graph preserving injections
- graph reversing maps, graph reversing injections

In other literature about orders, the distinction between the reflexive form, the irreflexive form and that which is not constrained to be either may not be made as above. Other authors may

- define orders using either the reflexive or the irreflexive form, ignoring the other possibilities

- call the reflexive form 'weak' and the irreflexive form 'strict', but again usually ignoring the possibility of being neither

[Hayes 1993, A.9] gives *partial_order X* for our reflexiveOrder X, and *total_order X* for our reflexiveTotalOrder X. [Halmos 1960] treats orders always as reflexive. [Suppes 1972] uses the word "strict" to distinguish the irreflexive case. [Enderton 1977] discusses the alternatives, then opts for an irreflexive definition.

There does seem to be good reason to name and to study the properties of both reflexive and irreflexive orders. For example, as we see in §27, a formal description of sorting can best be done using irreflexive orders. On the other hand, minima and maxima are defined more naturally with respect to orders that are reflexive.

[Spivey 1992, p104] uses the term 'monotonic' to refer to subset-preserving maps, and 'anti-monotonic' to refer to subset-reversing maps.

For further reading, [Davey & Priestly 1990] is a mathematical treatment of orders, bounds and lattices.

We use the following symbols for general orders, reflexive orders and irreflexive orders, respectively, in examples and definitions below.

$$\text{relation}\,(\_ \lessdot \_)$$
$$\text{relation}\,(\_ \preccurlyeq \_)$$
$$\text{relation}\,(\_ \prec \_)$$

If two elements are related by the order, they are *comparable*. If two elements are not related by the order, they are *incomparable*. For example, $\{a, b\}$ and $\{a, c\}$ are incomparable under the subset order.

Partial order

Intent

An order is a relation that is both transitive and antisymmetric.

Definition

orders:

$$\text{generic}\,(\text{order}\,\_)$$
$$\text{order}\,X == \text{transitive}\,X \cap \text{antisymmetric}\,X$$

reflexive orders:

Some orders have the property that all elements are related to themselves (as in 'subset of', 'less than *or equal to*'); they are *reflexive*.

generic (reflexiveOrder _)
reflexiveOrder X == order $X \cap$ reflexive X

irreflexive orders:

Some orders have the property that no elements are related to themselves (as in 'proper subset of', 'strictly less than'); they are *irreflexive*.

generic (irreflexiveOrder _)
irreflexiveOrder X == order $X \cap$ irreflexive X

In general, orders need be neither reflexive nor irreflexive, but in practice they tend to be one or the other.

Examples

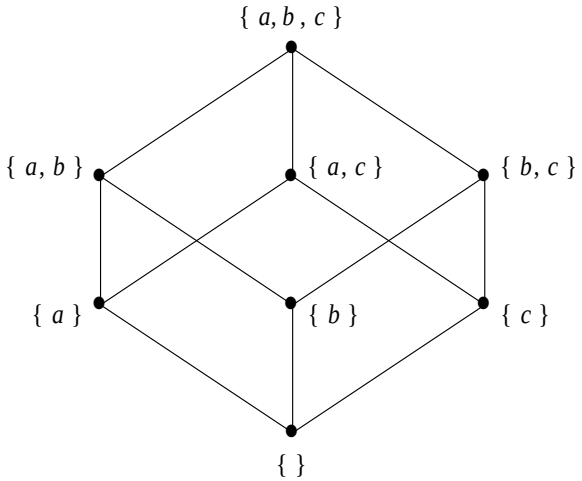


Figure 26.1 Example: the subsets of $\{a, b, c\}$ ordered by $\subseteq$.

1. $\varnothing[X^{2\times}] \in$ irreflexiveOrder X

2. $(\_ \subseteq \_)[X] \in \text{reflexiveOrder}\,\mathbb{P}\,X$
3. $(\_ \subset \_)[X] \in \text{irreflexiveOrder}\,\mathbb{P}\,X$
4. $\text{id}\,X \in \text{reflexiveOrder}\,X$. This is the trivial order, where each element is related (only) to itself. It is sometimes called an *anti-chain*.
5. The *Sort* relation uses irreflexive orders (§27).
6. An *interval* of real numbers can be modelled by its begining and end, $p = (x, y)$ where $x < y$. An order on intervals is $p \prec p' \Leftrightarrow x' < x \wedge y < y'$: the smaller interval is contained within the larger. It could be used to model *approximations* to a real number.
7. Another order on intervals is $p \prec p' \Leftrightarrow y < x'$: the earlier interval occurs entirely before the later. It could be used to model 'must finish before' order on time in a planning chart.
8. The *prefix* relation is a reflexive order on streams and sequences (§36). The *suffix* and *infix* relations are each reflexive orders on finite streams and finite sequences.
9. The *ancestor* relation is an irreflexive order (§42).

Laws

Law 26.1 The orders on a are a subset of the orders on the generic parameter set: order is subset-preserving.

$$[X]\ a : \mathbb{P}\,X \vdash \text{order}\,a \subseteq \text{order}\,X$$

Law 26.2 Adding reflexive pairs to an order results in an order.

$$[X]\ a : \mathbb{P}\,X;\ r : \text{order}\,X \vdash r \cup \text{id}\,a \in \text{order}\,X$$
$$[X]\ r : \text{order}\,X \vdash r \cup \text{id}\,X \in \text{reflexiveOrder}\,X$$

■ **Law 26.3** A subset of an order is antisymmetric, but need not be transitive. However, the following subsets do preserve transitivity, and hence order.

$$[X]\ a, b : \mathbb{P}\,X;\ r : \text{order}\,X \vdash a \lhd r \rhd b \in \text{order}\,X$$
$$[X]\ r, s : \text{order}\,X \vdash r \cap s \in \text{order}\,X$$
$$[X]\ a : \mathbb{P}\,X;\ r : \text{order}\,X \vdash r \setminus \text{id}\,a \in \text{order}\,X$$
$$[X]\ r : \text{order}\,X \vdash r \setminus \text{id}\,X \in \text{irreflexiveOrder}\,X$$

■ **Law 26.4** Transitive, irreflexive relations are irreflexive orders.

$$[X] \vdash \text{transitive}\, X \cap \text{irreflexive}\, X = \text{irreflexiveOrder}\, X$$

Law 26.5 Irreflexive orders are acyclic.

$$[X] \vdash \text{irreflexiveOrder}\, X \subseteq \text{acyclic}\, X$$

———————————————•

Poset

Intent

A poset is an ordered set with respect to a relation.

Definition

posets:

$$\text{poset}[X] == \lambda\, r : X \leftrightarrow X \bullet \{\ a : \mathbb{P}\, X \mid r \cap a^{2\times} \in \text{order}\, a\ \}$$

non-empty posets:

$$\text{poset}_1[X] == \lambda\, r : X \leftrightarrow X \bullet \text{poset}\, r \setminus \{\varnothing\}$$

If we want to refer to some set a that is ordered by some relation r, we declare it as $a : \text{poset}\, r$. The full relation r may or may not be an order itself.

Examples

- $\text{poset}\, \varnothing = \{\varnothing\}$
- $\text{poset}_1\, \varnothing = \varnothing$
- $\text{poset}(\text{id}\, X) = \mathbb{P}\, X$
- $\text{poset}(X^{2\times}) = \varnothing \cup \{\ x : X \bullet \{x\}\ \}$
- $\mathbb{R} \in \text{poset}(\_ < \_)$

Laws

Law 26.6 *poset* is a total surjection.

$$[X] \vdash \mathrm{poset}[X] \in (X \leftrightarrow X) \twoheadrightarrow \mathbb{P}\,\mathbb{P}\,X$$
$$[X] \vdash \mathrm{poset}_1[X] \in (X \leftrightarrow X) \twoheadrightarrow \mathbb{P}\,\mathbb{P}_1\,X$$

Law 26.7 If r is an order, any subset of X is a poset.

$$[X]\ r : \mathrm{order}\,X \vdash \mathrm{poset}\,r = \mathbb{P}\,X$$

Total order

Intent

If an order is such that *every* unequal pair is a member either of the order or of its inverse, it is a *total order*. There are no incomparable elements.

Definition

total orders:

$$\mathrm{generic}\,(\mathrm{totalOrder}\,\_)$$
$$\mathrm{totalOrder}\,X == \{\ r : \mathrm{order}\,X \mid r \cup r^{\sim} \cup \mathrm{id}\,X = X^{2\times}\ \}$$

reflexive total orders:

$$\mathrm{generic}\,(\mathrm{reflexiveTotalOrder}\,\_)$$
$$\mathrm{reflexiveTotalOrder}\,X == \mathrm{totalOrder}\,X \cap \mathrm{reflexive}\,X$$

irreflexive total orders:

$$\mathrm{generic}\,(\mathrm{irreflexiveTotalOrder}\,\_)$$
$$\mathrm{irreflexiveTotalOrder}\,X == \mathrm{totalOrder}\,X \cap \mathrm{irreflexive}\,X$$

Examples

1. The network diagram (section A.6.2) of a total order forms a single chain of elements drawn up the page.
2. The integers under 'less than' form a total order

$$\vdash (\_ \leq \_) \cap \mathbb{Z}^{2\times} \in \mathrm{reflexiveTotalOrder}\,\mathbb{Z}$$
$$\vdash (\_ < \_) \cap \mathbb{Z}^{2\times} \in \mathrm{irreflexiveTotalOrder}\,\mathbb{Z}$$

3. To introduce a new type with an order, but no additional structure (for example, none of the arithmetic structure that comes from using numbers).

$$[\mathit{TIME}]$$
$$\mathrm{relation}\,(\_\ \mathit{before}\ \_)$$

$$\mid\ \_\ \mathit{before}\ \_ : \mathrm{irreflexiveTotalOrder}\ \mathit{TIME}$$

Laws

Law 26.8 The complement and the inverse of a total order are both total orders, and identical to each other apart from in reflexivity

$$[X]\ r : \mathrm{totalOrder}\,X \vdash$$
$$(X \times X) \setminus r \in \mathrm{totalOrder}\,X$$
$$\land\ r^{\sim} \in \mathrm{totalOrder}\,X$$
$$\land\ ((X \times X) \setminus r) \cup \mathrm{id}\,X = r^{\sim} \cup \mathrm{id}\,X$$

Law 26.9 An irreflexive total order partitions its generic set into the elements before an element x, that element x, and the elements after x. (This motivates the names *predecessors* and *successors* for these relations.)

$$[X]\ r : \mathrm{irreflexiveTotalOrder}\,X;\ x : X \vdash$$
$$\langle \mathrm{predecessors}\,r\ x, \{x\}, \mathrm{successors}\,r\ x\rangle\ \mathrm{partition}\,X$$

•

Chain

Intent

A chain is a totally ordered set with respect to a relation. It is so called because of the form of the network diagram of a totally ordered set: a single line, or chain, of elements, unlike the branching diagram of figure A.5.

Definition

chains:

$$\mathrm{chain}[X] == \lambda\, r : X \leftrightarrow X \bullet \{\ a : \mathbb{P}\, X \mid r \cap a^{2\times} \in \mathrm{totalOrder}\ a\ \}$$

non-empty chains:

$$\mathrm{chain}_1[X] == \lambda\, r : X \leftrightarrow X \bullet \mathrm{chain}\, r \setminus \{\varnothing\}$$

If we want to refer to some set a that is totally ordered by some relation r, we declare it as $a : \mathrm{chain}\, r$. The full relation r may or may not be a total order itself.

Examples

1. $\mathrm{chain}\, \varnothing = \{\varnothing\}$
2. $\mathrm{chain}_1\, \varnothing = \varnothing$
3. $\mathrm{chain}(\mathrm{id}\, X) = \varnothing \cup \{\ x : X \bullet \{x\}\ \}$
4. $\mathrm{chain}(X^{2\times}) = \varnothing \cup \{\ x : X \bullet \{x\}\ \}$
5. $\mathbb{R} \in \mathrm{chain}(\_ < \_)$

Laws

Law 26.10 *chain* is a total surjection.

$$[X] \vdash \mathrm{chain}[X] \in (X \leftrightarrow X) \twoheadrightarrow \mathbb{P}\, \mathbb{P}\, X$$
$$[X] \vdash \mathrm{chain}_1[X] \in (X \leftrightarrow X) \twoheadrightarrow \mathbb{P}\, \mathbb{P}_1\, X$$

Law 26.11 If r is a total order, any subset of X is a chain.

$$[X]\ r : \text{totalOrder}\, X \vdash \text{chain}\, r = \mathbb{P}\, X$$

Law 26.12 A chain of r is also a poset of r.

$$[X]\ r : X \leftrightarrow X \vdash \text{chain}\, r \subseteq \text{poset}\, r$$

Preorder

Intent

A preorder is a relation that is reflexive and transitive.

Preorders capture the commonality between the two extremes of equivalence relations and orders: if a preorder is also symmetric, it is an equivalence relation (§24); if it is also antisymmetric, it is a (reflexive partial) order (§26).

Definition

$$\text{generic}\,(\text{preorder}\ \_)$$
$$\text{preorder}\, X == \text{reflexive}\, X \cap \text{transitive}\, X$$

Examples

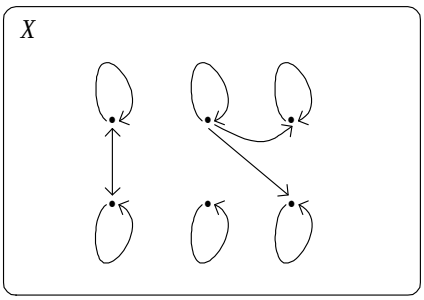


Figure 26.2 An example of a preorder.

1. $\{x \mapsto y, y \mapsto z, x \mapsto z, x \mapsto x, y \mapsto y, z \mapsto z\} \in \text{preorder}\{x, y, z\}$
2. $\{x \mapsto y, y \mapsto x, x \mapsto x, y \mapsto y\} \in \text{preorder}\{x, y\}$

3. $\mathrm{id}\, X \in \mathrm{preorder}\, X$
4. The relations *suffix* and *infix* are each preorders on streams and sequences (§36).
5. The refinement relation on implementations is a preorder. It is neither symmetric nor antisymmetric, for it is possible that both *a refines b* and *b refines a* where $a \neq b$. For example, the Phoenix and Apollo booking offices in [Woodcock & Davies 1996, §17.3] are mutual refinements, but are not equal.

Laws

Law 26.13 Preorder closure: the reflexive transitive closure of a relation is a preorder.

$$[X]\ r : X \leftrightarrow X \vdash r^* \in \mathrm{preorder}\, X$$

---•

A spectrum of orders

Intent

We have introduced a large number of sets of homogenous relations. We now clarify some of their interrelationships.

Opposites

Some of these relations are effectively opposites to each other, summed up in the laws:

$$[X] \mid X \neq \varnothing \vdash \mathrm{reflexive}\, X \cap \mathrm{irreflexive}\, X = \varnothing$$
$$[X] \vdash \mathrm{symmetric}\, X \cap \mathrm{antisymmetric}\, X = \mathbb{P}\, \mathrm{id}\, X$$
$$[X] \vdash \mathrm{transitive}\, X \cap \mathrm{intransitive}\, X = \{\ r : X \leftrightarrow X \mid \mathrm{dom}\, r \cap \mathrm{ran}\, r = \varnothing\ \}$$

Spectrum

We identify a "spectrum" of subsets of homogenous relations over X, all of which, assuming $X \neq \varnothing$, are non-empty and distinct, and give derived properties in the form of theorems. (The word "without" is used to draw attention to the fact that a certain condition has *not* been asserted, and also to imply that the set concerned is non-empty and non-trivial if the condition is denied.)

1. $r : \text{forest}\, X \vdash r \in \text{intransitive}\, X \cap \text{acyclic}\, X \cap \text{antisymmetric}\, X \cap \text{irreflexive}\, X$
2. $r : \text{intransitive}\, X \vdash r \in \text{acyclic}\, X \cap \text{antisymmetric}\, X \cap \text{irreflexive}\, X$
   (without $r \in X \nrightarrow X$)
3. $r : \text{acyclic}\, X \vdash r \in \text{antisymmetric}\, X \cap \text{irreflexive}\, X$
   (without $r \in \text{transitive}\, X$ or $r \in \text{intransitive}\, X$)
4. $r : \text{transitive}\, X \cap \text{irreflexive}\, X \vdash r \in \text{acyclic}\, X \cap \text{antisymmetric}\, X \cap \text{order}\, X$
5. $r : \text{transitive}\, X \cap \text{antisymmetric}\, X \vdash r \in \text{order}\, X$
   (without $r \in \text{reflexive}\, X$ or $r \in \text{irreflexive}\, X$)
6. $r : \text{transitive}\, X \cap \text{antisymmetric}\, X \cap \text{reflexive}\, X \vdash r \in \text{preorder}\, X \cap \text{order}\, X$
7. $r : \text{transitive}\, X \cap \text{reflexive}\, X \vdash r \in \text{preorder}\, X$
   (without $r \in \text{symmetric}\, X$ or $r \in \text{antisymmetric}\, X$)
8. $r : \text{transitive}\, X \cap \text{symmetric}\, X \vdash r \in \text{reflexive}\, X \cap \text{preorder}\, X \cap \text{equivalence}\, X$

Laws

Law 26.14 A preorder partitions its vertices into classes that are ordered under the original relation.

$$
\begin{array}{l}
[L, X]\ r : \text{preorder}\, X \vdash \\
\quad \exists f : L \nrightarrow \mathbb{P}\, X \bullet \\
\qquad f\ \text{partition}\, X \\
\qquad \land\ r \cap r^{\sim} = \bigcup \{\ a : \text{ran}\, f \bullet a^{2\times}\ \} \\
\qquad \land\ (\ \exists ro : \text{reflexiveOrder}(\text{ran}\, f) \bullet ro = \{\ a, b : \text{ran}\, f \mid a \times b \subseteq r\ \}\ )
\end{array}
$$

There are two special cases of this law.

- $r \in \text{symmetric}\, X \Rightarrow ro = \text{id}(\text{ran}\, f)$, the vacuous reflexive 'order'. r is an *equivalence relation*.
- $r \in \text{antisymmetric}\, X \Rightarrow ro = \{\ p : r \bullet \{p.1\} \mapsto \{p.2\}\ \}$, each element in the order ro is a singleton. r is an *order*.

Thus we see that, starting from any homogeneous relation r, we can form a preorder, and then partition the preorder into ordered sets.

- If r is strongly connected, r^* is the complete relation, the induced preorder partition is a single set, and the induced order is the identity pair on that single set.
- If r is acyclic, or an order, the induced preorder partition is a collection of singleton sets, and the induced order is the original order mapped over these singletons.
- Between these two extremes, this technique can be used to extract whatever order there is in a given relation.

Law 26.15 An alternative treatment to the above, briefer but perhaps less explicit, is given by

$$[X] \; r : \mathrm{transitive} \vdash \exists\, io : \mathrm{irreflexiveOrder}\, X \bullet io = r \setminus (r \cap r^{\sim})$$

In this case if r is strongly connected, the resultant order is the empty set, which is still a valid irreflexive order; it is the partial order which permits everything.

———————————•

Minimum and maximum

Intent

The minimum (maximum) element of a set with respect to a relation is defined as that unique member of the set, if any, that precedes (follows) all members of the set in the ordering implied by the relation.

The requirement that the minimum (maximum), if any, should be unique is guaranteed if the relation is antisymmetric (law 19.21) and many of the expected properties of minima and maxima as given in the laws below are predicated on that condition. Further properties depend on the relation's being an order, or a total order.

The straightforward formal definition implies that a minimum (maximum) cannot exist unless the relation is reflexive at least on the element concerned. In any practical application where a minimum (maximum) is required and reflexivity is in doubt, the reflexive closure of the first argument should therefore be used. (The alternative approach, of widening the definition to allow the reflexive case, was

rejected on the grounds that it would complicate the algebra and clutter the proofs to little or no advantage.)

Definition

minimum and maximum:

$$\begin{aligned}&\text{minimum}[X] == \\ &\qquad \lambda\, r : X \leftrightarrow X \bullet \{\ a : \mathbb{P}\, X;\ x : X \mid \{x\} = a \cap \text{lowerBound}\, r\ a\ \} \\ &\text{maximum}[X] == \\ &\qquad \lambda\, r : X \leftrightarrow X \bullet \{\ a : \mathbb{P}\, X;\ x : X \mid \{x\} = a \cap \text{upperBound}\, r\ a\ \}\end{aligned}$$

Examples

1. $\text{minimum}(\_ \subseteq \_)\{a, b, a \cap b, a \cup b\} = a \cap b$
2. $\text{minimum}(\_ \subseteq \_)\{\varnothing, a, b, a \cap b, a \cup b\} = \varnothing$
3. $\text{minimum}(\_ \leq \_)\{1, 2, 3\} = 1$
4. $\text{minimum}(\_ \geq \_)\{1, 2, 3\} = 3$
5. $\text{maximum}(\_ \subseteq \_)\{a, b, a \cap b, a \cup b\} = a \cup b$
6. $\text{maximum}(\_ \leq \_)\{1, 2, 3\} = 3$
7. $\text{maximum}(\_ \geq \_)\{1, 2, 3\} = 1$

Laws

Law 26.16 Laws about *maximum* are 'duals' of laws about *minimum*.

$$[X]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow X \vdash \text{maximum}(r^{\sim})a = \text{minimum}\, r\ a$$

Law 26.17 *minimum* is a total function of its relation argument. The resulting function is a partial function of its set argument: not all sets have a minimum element under a relation.

$$\begin{aligned}&[X]\ \vdash \text{minimum}[X] \in (X \leftrightarrow X) \to \mathbb{P}\, X \nrightarrow X \\ &[X]\ \vdash \text{maximum}[X] \in (X \leftrightarrow X) \to \mathbb{P}\, X \nrightarrow X\end{aligned}$$

Law 26.18 Minimum is insensitive to relational restriction outside its set argument. That is, if any one of minimum $r\ a$, or $\text{minimum}(a \lhd r)a$ or $\text{minimum}(r \rhd a)a$ is defined, they all are, and their values are equal.

$$[X]\ r : X \leftrightarrow X;\ a : \mathbb{P}\, X \vdash$$
$$a \lhd (\text{minimum}\, r) = a \lhd (\text{minimum}(a \lhd r)) = a \lhd (\text{minimum}(r \rhd a))$$

Law 26.19 An equivalent definition of *minimum*, not explicitly using *lowerBound*, is

$$[X]\ \vdash \text{minimum}[X] = \lambda\, r : X \leftrightarrow X \bullet$$
$$\{\ a : \mathbb{P}\, X;\ x : X \mid \{x\} = \{\ x : a \mid \forall\, y : a \bullet x \mapsto y \in r\ \}\ \}$$
$$[X]\ \vdash \text{maximum}[X] = \lambda\, r : X \leftrightarrow X \bullet$$
$$\{\ a : \mathbb{P}\, X;\ x : X \mid \{x\} = \{\ y : a \mid \forall\, x : a \bullet x \mapsto y \in r\ \}\ \}$$

Law 26.20 An equivalent definition of *minimum* for antisymmetric relations is

$$[X]\ r : \text{antisymmetric}\, X \vdash$$
$$\text{minimum}\, r = \{\ a : \mathbb{P}\, X;\ x : X \mid x \in a \subseteq \text{successors}\, r\ x\ \}$$
$$[X]\ r : \text{antisymmetric}\, X \vdash$$
$$\text{maximum}\, r = \{\ a : \mathbb{P}\, X;\ x : X \mid x \in a \subseteq \text{predecessors}\, r\ x\ \}$$

Law 26.21 An alternative and more explicit definition of *minimum* for antisymmetric relations is

$$[X]\ r : \text{antisymmetric}\, X \vdash$$
$$\text{minimum}\, r = \{\ a : \mathbb{P}\, X;\ x : X \mid x \in a \land (\ \forall\, y : a \bullet x \mapsto y \in r\ )\ \}$$
$$[X]\ r : \text{antisymmetric}\, X \vdash$$
$$\text{maximum}\, r = \{\ a : \mathbb{P}\, X;\ x : X \mid x \in a \land (\ \forall\, y : a \bullet y \mapsto x \in r\ )\ \}$$

Law 26.22 The minimum of a singleton set under a reflexive relation is that element.

$$[X]\ x : X;\ r : \text{reflexive}\, X \vdash \text{minimum}\, r\{x\} = x$$

Law 26.23 If a set and its subset both have a minimum, then the minimum of the set precedes the minimum of the subset.

$$[X]\ a, b : \mathbb{P}\, X;\ r : X \leftrightarrow X \mid \{a, b\} \subseteq \text{dom}(\text{minimum}\, r) \land b \subseteq a \vdash$$
$$\text{minimum}\, r\ a \mapsto \text{minimum}\, r\ b \in r$$

Law 26.24 If every set in a finite collection has a minimum under a total order, then so does their union, and it is the minimum of the individual minima.

$$[X]\ \alpha : \mathbb{F}(\mathbb{P}\, X);\ \_ \lessdot \_ : \text{totalOrder}\, X \mid$$
$$(\ \forall\, a : \alpha \bullet a \in \text{dom}(\text{minimum}(\_ \lessdot \_))\ ) \vdash$$
$$(\bigcup \alpha \mapsto \text{minimum}(\_ \lessdot \_)(\!|\ \alpha\ |\!)) \in \text{minimum}(\_ \lessdot \_)$$

This law does not necessarily hold for an infinite collection of sets. For example, a singleton set containing an integer has a minimum under the total order $\_ \leq \_$. The union of any finite set of these singletons has a minimum, but the union of all such sets is $\mathbb{Z}$, which has no minimum.

———————————————•

Greatest lower bound, least upper bound

Intent

The greatest lower (least upper) bound of a set with respect to a relation is the maximum of the lower bounds (minimum of the upper bounds).

Definition

greatest lower and least upper bounds:

$$\text{glb}[X] == \lambda\, r : X \leftrightarrow X \bullet (\text{maximum}\, r) \circ (\text{lowerBound}\, r)$$
$$\text{lub}[X] == \lambda\, r : X \leftrightarrow X \bullet (\text{minimum}\, r) \circ (\text{upperBound}\, r)$$

Examples

1. The bounds for the subset order are generalised intersection and union

$$[X]\ a : \mathbb{P}\, X \vdash \text{glb}(\_ \subseteq \_)[a] = \bigcap [a]$$
$$[X]\ a : \mathbb{P}\, X \vdash \text{lub}(\_ \subseteq \_)[a] = \bigcup [a]$$

2. The greatest lower bound of the numerical order on natural numbers is the *min* function: $\text{glb}((\_ \leq \_) \cap \mathbb{N}^{2\times}) \subseteq \text{min}$. *min* is total on all non-empty subsets of $\mathbb{N}$.
3. The least upper bound of the numerical order on natural numbers is the *max* function: $\text{lub}((\_ \leq \_) \cap \mathbb{N}^{2\times}) \subseteq \text{max}$. *max* is total only on finite non-empty subsets of $\mathbb{N}$.

4. The greatest lower (least upper) bound need not be a member of the set

$$\mathrm{glb}((\_ \leq \_) \cap (\mathbb{R} \cap \mathbb{R}))\{\ x : \mathbb{R} \mid 0 < x\ \} = 0$$
$$\mathrm{lub}(\_ \subseteq \_)\{\{a\}, \{b\}\} = \{a, b\}$$

5. The greatest lower bound of the prefix order on sequences is 'longest common prefix', which is total on all non-empty sets of sequences.

Laws

Law 26.25 Laws about *lub* are duals of laws about *glb*.

$$[X]\ a : \mathbb{P}\, X;\ r : X \leftrightarrow X \vdash \mathrm{lub}(r^{\sim})\, a = \mathrm{glb}\, r\ a$$

Law 26.26 *glb* is a total function.

$$[X]\ \vdash \mathrm{glb}[X] \in (X \leftrightarrow X) \to \mathbb{P}\, X \nrightarrow X$$
$$[X]\ \vdash \mathrm{lub}[X] \in (X \leftrightarrow X) \to \mathbb{P}\, X \nrightarrow X$$

Law 26.27 If an antisymmetric set has a minimum, that minimum is also the greatest lower bound.

$$[X]\ r : \mathrm{antisymmetric}\, X \vdash \mathrm{minimum}\, r \subseteq \mathrm{glb}\, r$$

Law 26.28 If a set has a greatest lower bound, all other lower bounds precede it. (This follows immediately from the definitions.)

$$[X]\ r : X \leftrightarrow X;\ x : X;\ a : \mathbb{P}_1\, X \mid a \in \mathrm{dom}(\mathrm{glb}\, r) \land x \in \mathrm{lowerBound}\, r\ a \vdash x \mapsto \mathrm{glb}\, r\ a \in r$$

Law 26.29 If a set has a greatest lower bound with respect to an order, any value that precedes it is also a lower bound.

$$[X]\ r : \mathrm{order}\, X;\ x : X;\ a : \mathbb{P}_1\, X \mid a \in \mathrm{dom}(\mathrm{glb}\, r) \land x \mapsto \mathrm{glb}\, ra \in r \vdash x \in \mathrm{lowerBound}\, r\ a$$

---•

Well order

Intent

A *well ordering* is a reflexive order where every non-empty subset of the elements has a minimum with respect to the order. It has a 'first element' for the order.

Definition

well orders:

$$\begin{array}{l} \text{generic}\,(\text{wellOrder}\,\_) \\ \text{wellOrder}\,X == \{\ rto : \text{reflexiveOrder}\,X \mid \text{dom}(\text{minimum}\,rto) = \mathbb{P}_1\,X\ \} \end{array}$$

Examples

1. The natural numbers under $\leq$ form a well order

$$\vdash (\_ \leq \_) \cap \mathbb{N}^{2\times} \in \text{wellOrder}\,\mathbb{N}$$

2. $1 \prec 3 \prec 5 \prec 7 \prec \ldots \prec 0 \prec 2 \prec 4 \prec 6 \prec \ldots$ is also a well order on $\mathbb{N}$.
3. $(\_ \leq \_)$ restricted to the integers, $\mathbb{Z}^{2\times}$, is not a well order, because there is no minimum element. For a similar reason, the relation $(\_ \geq \_)$ on the naturals is not a well order.
4. $0 \prec 1 \prec$ -1 $\prec 2 \prec$ -2 $\prec 3 \prec$ -3 $\prec \ldots$ is a well order on $\mathbb{Z}$.
5. $(\_ \leq \_)$ restricted to non-negative reals, $\mathbb{R}_+^{2\times}$, is not a well order, because not every subset has a minimum (consider the subset of positive reals, $\mathbb{R}_+$).

Laws

Law 26.30 Well orders are total orders.

$$[X] \ \vdash \text{wellOrder}\,X \subseteq \text{reflexiveTotalOrder}\,X$$

Law 26.31 Any finite reflexive total order is a well order.

$$[X]\ r : (X \nrightarrow X) \cap \text{reflexiveTotalOrder}\,X \vdash r \in \text{wellOrder}\,X$$

Law 26.32 *Well-ordering principle*: any set can be well ordered. (Such a well order might not be the "usual" order.)

$$[X] \vdash \exists\, wo : \mathrm{wellOrder}\, X \bullet true$$

The well-ordering principle is equivalent to the axiom of choice. See, for example [Halmos 1960, chapter 17], [Suppes 1972, chapter 8], [Enderton 1977, chapter 7].

Law 26.33 A well ordering implies an induction principle.

$$[X]\ wo : wellOrderX;\ a, b;\ \mathbb{P}\, X \mid (\ \forall\, x : a \mid \mathrm{predecessors}\, wo\ x \setminus \{x\} \subseteq b \bullet x \in b\ ) \vdash a \subseteq b$$

This form of induction is sometimes know as "transfinite induction", because it can be applied to well orderings that are not enumerable (§35). In practice it is often usefully applied to orderings that are in fact enumerable.

———————————•

Well founded chain

Intent

A well founded chain is a well ordered set with respect to a relation.

Definition

well founded chains:

$$\mathrm{wellFoundedChain}[X] == \lambda\, r : X \leftrightarrow X \bullet \{\ a : \mathbb{P}\, X \mid r \cap a^{2\times} \in \mathrm{wellOrder}\, a\ \}$$

non-empty well founded chains:

$$\mathrm{wellFoundedChain}_1[X] == \lambda\, r : X \leftrightarrow X \bullet \mathrm{wellFoundedChain}\, r \setminus \{\varnothing\}$$

If we want to refer to some set a that is well ordered by some relation r, we declare it as $a : \mathrm{wellFoundedChain}\, r$. The full relation r may or may not be a well order itself.

Laws

Law 26.34 *wellFoundedChain* is a total function.

$$[X] \vdash \text{wellFoundedChain}[X] \in (X \leftrightarrow X) \to \mathbb{P}\,\mathbb{P}\,X$$
$$[X] \vdash \text{wellFoundedChain}_1[X] \in (X \leftrightarrow X) \to \mathbb{P}\,\mathbb{P}_1\,X$$

Law 26.35 If r is a well order, any subset of X is a well founded chain.

$$[X]\ r : \text{wellOrder}\,X \vdash \text{wellFoundedChain}\,r = \mathbb{P}\,X$$

•

Graph-preserving maps

Intent

We defined two kinds of *morphisms* in §21: mappings that maintain the structure of functions. Now we define further kinds of structure maintaining maps: ones that maintain a relation.

If we have relations on two sets, and a function that maps elements of one set to the other, then it is *graph-preserving* if elements from the first set that are in its relation remain related when transformed.

Definition

graph-preserving maps:

GraphPreservingMap $[X, Y]$

$$r_x : X \leftrightarrow X$$
$$r_y : Y \leftrightarrow Y$$
$$f : X \nrightarrow Y$$

$$\forall\, x, y : \text{dom}\, f \mid x \mapsto y \in r_x \bullet f\ x \mapsto f\ y \in r_y$$

GraphPreservingInjection $[X, Y]$

GraphPreservingMap$[X, Y]$

$$f \in X \rightarrowtail Y$$

Graph-preserving maps are of interest, because they allow us to 'lift' some relation on a pair of elements to some relation on the image of those elements under the map. See the examples below. There are three distinct cases:

1. $X = Y$ and $r_x = r_y$: the same relation over the same type.
2. $X \neq Y$ and $r_x = r_y$: the same generic relation over different types. For example, subset order on sets $\mathbb{P}\, X$, lifted to subset order on sets of sets $\mathbb{P}\,\mathbb{P}\, X$.
3. $r_x \neq r_y$: two different relations (over the same or different types).

Examples

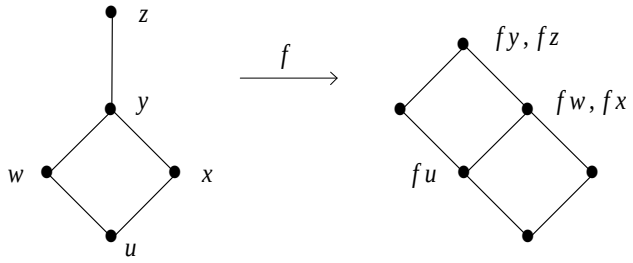


Figure 26.3 Network diagram example of a graph-preserving map on an order

1. Cartesian product is subset-preserving on both arguments.
2. Set difference is subset-preserving on its first argument.
3. Domain and range are subset-preserving.
4. Transitive closure is subset-preserving.
5. Addition is numerical order-preserving on both arguments.
6. Subtraction is numerical order-preserving on its first argument.
7. Sequence concatenation is prefix order-preserving on its second argument.
8. Sequence concatenation is suffix order-preserving on its first argument.

Laws

■ **Law 26.36** A graph-preserving map applied to the *glb* and *lub* of a set gives wider bounds than the *glb* and *lub* of the image of the set through the map, where

all such bounds exist.

$$
\begin{array}{l}
[X, Y]\ a : \mathbb{P}\, X;\ \mathrm{GraphPreservingMap}[X, Y] \mid \\
\qquad a \in \mathrm{dom}(\mathrm{glb}\, r_x) \\
\qquad \land\ f(\!|\ a\ |\!) \in \mathrm{dom}(\mathrm{glb}\, r_y) \\
\qquad \land\ \mathrm{glb}\, r_x\ a \in \mathrm{dom}\, f \vdash \\
\quad f(\mathrm{glb}\, r_x\ a) \mapsto \mathrm{glb}\, r_y(f(\!|\ a\ |\!)) \in r_y
\end{array}
$$

$$
\begin{array}{l}
[X, Y]\ a : \mathbb{P}\, X;\ \mathrm{GraphPreservingMap}[X, Y] \mid \\
\qquad a \in \mathrm{dom}(\mathrm{lub}\, r_x) \\
\qquad \land\ f(\!|\ a\ |\!) \in \mathrm{dom}(\mathrm{lub}\, r_y) \\
\qquad \land\ \mathrm{lub}\, r_x\ a \in \mathrm{dom}\, f \vdash \\
\quad \mathrm{lub}\, r_y(f(\!|\ a\ |\!)) \mapsto f(\mathrm{lub}\, r_x\ a) \in r_y
\end{array}
$$

One common specialisation is when both relations r_x and r_y are the subset relation, and hence the bounds are union and intersection.

$$
\begin{array}{l}
[X, Y]\ \alpha : \mathbb{P}\, X;\ f : \mathbb{P}\, X \nrightarrow \mathbb{P}\, Y \mid \\
\qquad \bigcap \alpha \in \mathrm{dom}\, f \\
\qquad \land\ (\ \forall\, a, b : \mathrm{dom}\, f \mid a \subseteq b \bullet f\ a \subseteq f\ b\ ) \vdash \\
\quad f(\bigcap \alpha) \subseteq \bigcap (f(\!|\ \alpha\ |\!)) \\
[X, Y]\ \alpha : \mathbb{P}\, X;\ f : \mathbb{P}\, X \nrightarrow \mathbb{P}\, Y \mid \\
\qquad \bigcup \alpha \in \mathrm{dom}\, f \\
\qquad \land\ (\ \forall\, a, b : \mathrm{dom}\, f \mid a \subseteq b \bullet f\ a \subseteq f\ b\ ) \vdash \\
\quad \bigcup (f(\!|\ \alpha\ |\!)) \subseteq f(\bigcup \alpha)
\end{array}
$$

■ **Law 26.37** Any function that distributes through least upper bound or greatest lower bound is graph-preserving.

$$
\begin{array}{l}
[X, Y]\ r_x : \mathrm{reflexive}\, X;\ r_y : \mathrm{reflexive}\, Y;\ f : X \nrightarrow Y \mid \\
\qquad \forall\, a : \mathbb{P}\, X \mid a \in \mathrm{dom}(\mathrm{lub}\, r_x) \land f(\!|\ a\ |\!) \in \mathrm{dom}(\mathrm{lub}\, r_y) \land \mathrm{lub}\, r_x\ a \in \mathrm{dom}\, f \bullet \\
\qquad\quad f(\mathrm{lub}\, r_x\ a) = \mathrm{lub}\, r_y(f(\!|\ a\ |\!)) \vdash \\
\quad \mathrm{GraphPreservingMap}[X, Y]
\end{array}
$$

$$
\begin{array}{l}
[X, Y]\ r_x : \mathrm{reflexive}\, X;\ r_y : \mathrm{reflexive}\, Y;\ f : X \nrightarrow Y \mid \\
\qquad \forall\, a : \mathbb{P}\, X \mid a \in \mathrm{dom}(\mathrm{glb}\, r_x) \land f(\!|\ a\ |\!) \in \mathrm{dom}(\mathrm{glb}\, r_y) \land \mathrm{glb}\, r_x\ a \in \mathrm{dom}\, f \bullet \\
\qquad\quad f(\mathrm{glb}\, r_x\ a) = \mathrm{glb}\, r_y(f(\!|\ a\ |\!)) \vdash \\
\quad \mathrm{GraphPreservingMap}[X, Y]
\end{array}
$$

One common specialisation is when both relations r_x and r_y are the subset relation, and hence the bounds are union and intersection.

$$[X, Y]\ f : \mathbb{P}\, X \pfun \mathbb{P}\, Y \mid \forall\, \alpha : \mathbb{P}\, \mathbb{P}\, X \mid \bigcup \alpha \in \operatorname{dom} f \bullet f(\bigcup \alpha) = \bigcup(f(\!|\ \alpha\ |\!)) \vdash \\ \forall\, a, b : \operatorname{dom} f \mid a \subseteq b \bullet f\ a \subseteq f\ b$$

$$[X, Y]\ f : \mathbb{P}\, X \pfun \mathbb{P}\, Y \mid \forall\, \alpha : \mathbb{P}\, \mathbb{P}\, X \mid \bigcap \alpha \in \operatorname{dom} f \bullet f(\bigcap \alpha) = \bigcap(f(\!|\ \alpha\ |\!)) \vdash \\ \forall\, a, b : \operatorname{dom} f \mid a \subseteq b \bullet f\ a \subseteq f\ b$$

This law shows that the existence of a distributive law is a stronger property than order preservation. (In practice, a stronger law may hold, where the implication holds both ways: $a \subseteq b \Leftrightarrow f\ a \subseteq f\ b$.)

•

Graph-reversing maps

Intent

If we have relations on two sets, and a function that maps elements of one set to the other, then it is *graph-reversing* if elements from the first set that are in its relation are related oppositely when transformed.

Definition

graph-reversing maps:

GraphReversingMap $[X, Y]$

$$r_x : X \leftrightarrow X \\ r_y : Y \leftrightarrow Y \\ f : X \pfun Y$$

$$\forall\, x, y : \operatorname{dom} f \mid x \mapsto y \in r_x \bullet f\ y \mapsto f\ x \in r_y$$

GraphReversingInjection $[X, Y]$

$$\text{GraphReversingMap}[X, Y]$$

$$f \in X \pinj Y$$

Examples

1. Generalised intersection is subset-reversing.
2. Set difference is subset-reversing on its second argument.
3. Subtraction is numerical order-reversing on its second argument.
4. *min* and *max* are numerical order-reversing.

Laws

Law 26.38 A graph-reversing map on r_x and r_y can be transformed into a graph-preserving map on r_x and ${r_y}^\sim$.

$$[X, Y]\ \text{GraphReversingMap}[X, Y] \vdash \\ \quad \text{GraphPreservingMap}[X, Y]\ {}_1 \bullet f = f_1 \land r_x = r_{x1} \land r_y = {r_{y1}}^\sim$$

However, we define the separate case, to help expose the pseudo-distributive properties when the relations are the same.

Law 26.39 A graph-reversing map applied to the *glb* and *lub* of a set gives wider bounds than the *lub* and *glb* of the image of the set through the map, where all such bounds exist.

$$[X, Y]\ a : \mathbb{P}\, X;\ \text{GraphReversingMap}[X, Y] \mid \\ \qquad a \in \text{dom}(\text{lub}\, r_x) \\ \qquad \land\ f(\!|\ a\ |\!) \in \text{dom}(\text{glb}\, r_y) \\ \qquad \land\ \text{lub}\, r_x\ a \in \text{dom}\, f \vdash \\ \quad f(\text{lub}\, r_x\ a) \mapsto \text{glb}\, r_y(f(\!|\ a\ |\!)) \in r_y$$

$$[X, Y]\ a : \mathbb{P}\, X;\ \text{GraphReversingMap}[X, Y] \mid \\ \qquad a \in \text{dom}(\text{glb}\, r_x) \\ \qquad \land\ f(\!|\ a\ |\!) \in \text{dom}(\text{lub}\, r_y) \\ \qquad \land\ \text{glb}\, r_x\ a \in \text{dom}\, f \vdash \\ \quad \text{lub}\, r_y(f(\!|\ a\ |\!)) \mapsto f(\text{glb}\, r_x\ a) \in r_y$$

One common specialisation is when both relations r_x and r_y are the subset relation, and hence the bounds are union and intersection.

$$[X, Y]\ \alpha : \mathbb{P}\, X;\ f : \mathbb{P}\, X \nrightarrow \mathbb{P}\, Y \mid \\ \qquad \bigcup \alpha \in \text{dom}\, f \\ \qquad \land\ (\ \forall\, a, b : \text{dom}\, f \mid a \subseteq b \bullet f\ b \subseteq f\ a\ ) \vdash \\ \quad f(\bigcup \alpha) \subseteq \bigcap (f(\!|\ \alpha\ |\!))$$

$$[X, Y]\ \alpha : \mathbb{P}\, X;\ f : \mathbb{P}\, X \pfun \mathbb{P}\, Y \mid \bigcap \alpha \in \operatorname{dom} f \land (\ \forall\, a, b : \operatorname{dom} f \mid a \subseteq b \bullet f\ b \subseteq f\ a\ ) \vdash \bigcup (f(\!|\ \alpha\ |\!)) \subseteq f(\bigcap \alpha)$$

Law 26.40 Any function that pseudo-distributes through bounds is order-reversing.

$$[X, Y]\ r_x : \operatorname{reflexive} X;\ r_y : \operatorname{reflexive} Y;\ f : X \pfun Y \mid \forall\, a : \mathbb{P}\, X \mid a \in \operatorname{dom}(\operatorname{lub} r_x) \land f(\!|\ a\ |\!) \in \operatorname{dom}(\operatorname{glb} r_y) \land \operatorname{lub} r_x\ a \in \operatorname{dom} f \bullet f(\operatorname{lub} r_x\ a) = \operatorname{glb} r_y(f(\!|\ a\ |\!)) \vdash \operatorname{GraphReversingMap}[X, Y]$$

$$[X, Y]\ r_x : \operatorname{reflexive} X;\ r_y : \operatorname{reflexive} Y;\ f : X \pfun Y \mid \forall\, a : \mathbb{P}\, X \mid a \in \operatorname{dom}(\operatorname{glb} r_x) \land f(\!|\ a\ |\!) \in \operatorname{dom}(\operatorname{lub} r_y) \land \operatorname{glb} r_x\ a \in \operatorname{dom} f \bullet \bullet f(\operatorname{glb} r_x\ a) = \operatorname{lub} r_y(f(\!|\ a\ |\!)) \vdash \operatorname{GraphReversingMap}[X, Y]$$

One common specialisation is when both relations r_x and r_y are the subset relation, and hence the bounds are union and intersection.

$$[X, Y]\ f : \mathbb{P}\, X \pfun \mathbb{P}\, Y \mid \forall\, \alpha : \mathbb{P}\, \mathbb{P}\, X \mid \bigcup \alpha \in \operatorname{dom} f \bullet f(\bigcup \alpha) = \bigcap (f(\!|\ \alpha\ |\!)) \vdash \forall\, a, b : \operatorname{dom} f \mid a \subseteq b \bullet f\ b \subseteq f\ a$$

$$[X, Y]\ f : \mathbb{P}\, X \pfun \mathbb{P}\, Y \mid \forall\, \alpha : \mathbb{P}\, \mathbb{P}\, X \mid \bigcap \alpha \in \operatorname{dom} f \bullet f(\bigcap \alpha) = \bigcup (f(\!|\ \alpha\ |\!)) \vdash \forall\, a, b : \operatorname{dom} f \mid a \subseteq b \bullet f\ b \subseteq f\ a$$

This law shows that the existence of a pseudo-distributive law is a stronger property than order reversal. (In practice, a stronger law may hold, where the implication holds both ways: $a \subseteq b \Leftrightarrow f\ b \subseteq f\ a$.)

———————————————•

Chapter 27

Sorting

Sort

Intent

A specification of sorting items can be undertaken by using orders, and enables us to see the underlying structure clearly. Because we have abstracted out the concept of an order, we can clearly see that

- we do not need to use numbers to define the resulting order after sorting; any ordered set will do
- we do not need to have a total order defined on the items to be sorted; a partial order will do

The operation of methodically sorting records into order is probably at least as old as writing. For example, in a dictionary or other word-list the entries are usually sorted into alphabetical order; in recording events that have occurred or are planned to occur it is often best to sort in order of time; in modern recording systems it is common to have some kind of reference number and to display records in that numeric order.

Overview

The operation of sorting starts with the input, which is some labelled set of records. We want to sort the records into some order; the labelling serves to allow the same record to occur more than once. If the input is presented as a sequence, the labelling set is $\mathbb{N}_1$, but in general any labelling set will do.

We refer to the record type as $[X]$ and the input labelling set as $[I]$. We declare the input as

$$in : I \pfun X$$

If the input is instead declared as an unlabelled set, we can easily assimilate it to our model by making it self-labelling. To do that we equate I with X, and if we have a set

$$inSet : \mathbb{P}\, X$$

we say

$$in = \mathrm{id}\, inSet$$

To sort, there must be an order on X into which the records are to be sorted. In data processing contexts this is often controlled by some numeric or alphabetic comparison on one or several key fields that form part of the record. We might on the other hand be sorting an acyclic graph where the directed links are given explicitly; this is sometimes referred to as a *topological sort*.

There are no grounds for requiring or expecting the order to be *total* (although some published formal treatments of this subject have made that assumption). Some distinct pairs of values of X may not appear either way round in the order; these are *incomparable* pairs. In performing our sort, however, we wish to treat such incomparable pairs on the same footing as pairs where two distinct labels identify the same value of X; that is, to require pairs like (x, x) also to be incomparable. So we want the order to be *irreflexive*.

$$xOrder : \mathrm{irreflexiveOrder}\, X$$

In practice both X and $xOrder$ may well be quite complicated in structure, but we have abstracted away from that complexity.

If a sorting order $rawOrd$ is given that is not irreflexive, it is straightforward to define a corresponding order that is, by taking the *irreflexive residue*:

$$xOrder = rawOrd \setminus \mathrm{id}\, X$$

The purpose of the sort is to define an output ordering of the records. This means putting them in a total order. So with output index $[J]$ we declare

$$jOrder : \mathrm{irreflexiveTotalOrder}\, J$$
$$out : J \nrightarrow X$$

Quite often J is $\mathbb{N}_1$, $jOrder$ is $\_ < \_$, and $\mathrm{dom}\, out$ is $1 \mathinner{..} n$ for some n. However, J need not be numeric.

To define our sort we need to stipulate that the output records are the same as the input records. We do this by declaring a permutation of the indices

$$perm : J \rightarrowtail\!\!\!\!\!\!\rightarrow I$$

that is a bijection on the domains

$$perm \in \operatorname{dom} out \rightarrowtail\!\!\!\!\!\!\twoheadrightarrow \operatorname{dom} in$$

and governs the relationship between the output and the input

$$out = perm \fatsemi in$$

We constrain the output to be in the desired order

$$\forall p, q : out \mid p.2 \mapsto q.2 \in xOrder \bullet p.1 \mapsto q.1 \in jOrder$$

Definition

The complete definition of the sorting process is:

Sort $[I, J, X]$

$$in : I \pfun X$$
$$xOrder : \operatorname{irreflexiveOrder} X$$
$$jOrder : \operatorname{irreflexiveTotalOrder} J$$
$$out : J \pfun X$$
$$perm : J \rightarrowtail\!\!\!\!\!\!\rightarrow I$$

$$perm \in \operatorname{dom} out \rightarrowtail\!\!\!\!\!\!\twoheadrightarrow \operatorname{dom} in$$
$$out = perm \fatsemi in$$
$$\forall p, q : out \mid p.2 \mapsto q.2 \in xOrder \bullet p.1 \mapsto q.1 \in jOrder$$

Since *xOrder* need not be total in general, there may be pairs of input X values that are incomparable, and the above specification allows them to be output in either order. Hence the process specified by *Sort* is nondeterministic, and many different outputs may be permitted.

Stable sorting, an option often offered by commercial sorting packages, has the additional constraint that incomparable records are placed in the output in the

same order as they were in the input. To describe this we need to recognise the order of the input index.

$$
\begin{array}{|l}
\hline
\text{StableSort}\,[I, J, X] \\
\text{Sort}[I, J, X] \\
iOrder : \text{irreflexiveTotalOrder}\, I \\
\hline
\forall\, p, q : out \mid p.2 \mapsto q.2 \notin \text{symmetricClosure}\, xOrder \bullet \\
\qquad p.1 \mapsto q.1 \in jOrder \Leftrightarrow perm\ p.1 \mapsto perm\ q.1 \in iOrder \\
\hline
\end{array}
$$

Stable sorting is deterministic.

$\_\_\_\_\_\bullet$

Chapter 28

Two binary operators

We define structures that have two binary operators over a set.

- rings
- integral domains
- fields
- ordered integral domains
- ordered fields
- complete fields

The main motivation is to give sufficient machinery to define numbers, §29. However, these standard mathematical constructs can find a wider use.

Ring

Intent

A ring has two binary operators, one called $\dot{+}$ forming an *additive* Abelian group (hence with an additive identity element, 'zero', and an additive inverse, 'negation'), and one called $\dot{*}$ forming a *multiplicative* semigroup. The operators are linked by the distributive property on both sides.

Continuing the numerical analogy, we write the additive inverse, negation, to resemble unary minus, and make it right associative, so that $\dot{-}\,\dot{-}\,x = \dot{-}(\dot{-}\,x)$, as expected. (Without an operator template definition, we would instead have the usual left-associative function application $\dot{-}\,\dot{-}\,x = (\dot{-}\,\dot{-})x$.)

Definition

function 30 leftassoc $(\_ \dot{+} \_)$

function 40 leftassoc $(\_ \dot{*} \_)$

function $(\dot{-}\ \_)$

Ring $[X]$

AbelianGroup$[X][_ \dot{+} _/_ \diamond _, O/e, \dot{-}\ _/inv]$
SemiGroup$[X][_ \dot{*} _/_ \diamond _]$

$$\forall\, x, y, z : g \bullet$$
$$x \dot{*} (y \dot{+} z) = (x \dot{*} y) \dot{+} (x \dot{*} z)$$
$$\land\ (y \dot{+} z) \dot{*} x = (y \dot{*} x) \dot{+} (z \dot{*} x)$$

Examples

1. The terms 'additive' and 'multiplicative' are used for the ring operators because this is what they correpond to in the ring of integers. But not all rings are numerical. A *Boolean ring* has $x \dot{*} x = x$ and $x \dot{+} x = O$.

$$\mathit{BooleanRing}[X] == [\ \text{Ring}[X] \mid \forall\, x : g \bullet x \dot{*} x = x \land x \dot{+} x = O\ ]$$

2. Symmetric set difference (as the additive operator) and intersection (as the multiplicative operator) form a Boolean ring over sets. The additive identity is the empty set; the additive inverse is the identity relation.

$$[X] \vdash$$
$$\exists\, \mathit{BooleanRing}[\mathbb{P}\, X] \bullet$$
$$g = \mathbb{P}\, X \land (\_ \dot{+} \_) = (\_ \ominus \_) \land O = \varnothing$$
$$\land\ (\dot{-}\ \_) = \mathrm{id}(\mathbb{P}\, X) \land (\_ \dot{*} \_) = (\_ \cap \_)$$

3. If R is a ring, then the set of $n^{2\times}$ matrices over R, under the usual matrix addition and multiplication, is also a ring.
4. If R is a ring, then the set of polynomials with coefficients drawn from R, under polynomial addition and multiplication, is also a ring.

Laws

■ **Law 28.1** The additive identity element acts as a zero for multiplication.

$$[X]\ \mathrm{Ring}[X];\ x : X \mid x \in g \vdash x \dot{*} O = O = O \dot{*} x$$

■ **Law 28.2** The additive inverse distributes through multiplication.

$$[X]\ \mathrm{Ring}[X];\ x, y : X \mid \{x, y\} \subseteq g \vdash x \dot{*} \dot{-} y = \dot{-}(x \dot{*} y) = \dot{-} x \dot{*} y$$

■ **Law 28.3** Addition of additive inverse behaves like a binary subtraction operator.

$$\mathrm{function}\ 30\ \mathrm{leftassoc}\ (\_ \dot{-} \_)$$

$$\begin{array}{l}[X]\ \mathrm{Ring}[X];\ x, y, z : X;\ \_ \dot{-} \_ : X^{2\times} \nrightarrow X \mid \\ \qquad \{x, y, z\} \subseteq g \\ \qquad \land\ g^{2\times} \lhd (\_ \dot{-} \_) \in g^{2\times} \to g \\ \qquad \land\ (\ \forall x, y : g \bullet x \dot{-} y = x \dot{+} \dot{-} y\ ) \vdash \\ x \dot{*} (y \dot{-} z) = x \dot{*} y \dot{-} x \dot{*} z \\ \land\ (x \dot{-} y) \dot{*} z = x \dot{*} z \dot{-} y \dot{*} z\end{array}$$

The above three laws provide further justification for the choice of the 'arithmetical' names of the two ring operators.

■ **Law 28.4** If every element x in a ring obeys $x \dot{*} x = x$, then it is a Boolean ring.

$$[X]\ \mathrm{Ring}[X] \mid \forall x : g \mid x \dot{*} x = x \vdash \mathit{BooleanRing}[X]$$

•

Integral domain

Intent

An integral domain is a ring where the multiplicative operator forms an abelian monoid (hence with a multiplicative identity element, 'unity'), and there are no non-zero factorisations of zero (that is, if two elements multiply to zero, at least one of them is zero).

Definition

IntegralDomain $[X]$

$\mathrm{Ring}[X]$

$\mathrm{AbelianMonoid}[X][\_ \dot{*} \_ / \_ \diamond \_, I / e]$

$O \neq I$

$\forall\, x, y : g \bullet x \dot{*} y = O \Leftrightarrow x = O \vee y = O$

Examples

1. The integers under addition and multiplication modulo n form a (finite) integral domain.

$$[X]\ n : \mathbb{N} \vdash \exists\, \mathrm{IntegralDomain}[\mathbb{A}] \bullet g = 0 \mathinner{..} n - 1 \wedge (\_ \dot{+} \_) = (\_ + \_) \bmod n \wedge O = 0 \wedge (\dot{-}\_) = (\ \lambda\, m : g \bullet \text{-}m \bmod n\ ) \wedge (\_ \dot{*} \_) = (\_ * \_) \bmod n \wedge I = 1$$

2. The integers under addition and multiplication form an (infinite) integral domain.

$$\vdash \exists\, \mathrm{IntegralDomain}[\mathbb{A}] \bullet g = \mathbb{N} \wedge (\_ \dot{+} \_) = (\_ + \_) \wedge O = 0 \wedge (\dot{-}\_) = (\text{-}\_) \wedge (\_ \dot{*} \_) = (\_ * \_) \wedge I = 1$$

•

Field

Intent

A field is an integral domain where the non-zero elements form a group under the multiplicative operator (hence with a multiplicative inverse operation, 'reciprocal', on the non-zero elements).

Definition

function $(\_^{\dot{-}1})$

Field $[X]$

$\text{IntegralDomain}[X]$

$\text{AbelianGroup}[X][g_0/g, \_ \dot{*} \_ / \_ \diamond \_, I/e, \_^{\dot{-}1}/inv]$

$g_0 = g \setminus \{O\}$

Examples

1. The integers under addition and multiplication modulo p, where p is prime, form a (finite) field.

$$p : \text{prime} \vdash \exists\, \text{Field}[\mathbb{A}] \bullet g = 0 \mathinner{..} p - 1 \land (\_ \dot{+} \_) = (\_ + \_) \bmod p \land O = 0 \land (\_ \dot{*} \_) = (\_ * \_) \bmod p \land I = 1$$

Laws

Law 28.5 The size of a finite field is a power of a prime. (See, for example, [Stewart 1973, §17].)

$$[X] \;\; \text{Field}[X] \mid \text{finite}\, g \vdash \exists\, n : \mathbb{N}_1;\; p : \text{prime} \bullet \#g = p \mathbin{**} n$$

Law 28.6 Two finite fields of the same size are isomorphic.

$$[X, Y]\ \text{Field}[X];\ \text{Field}[Y]' \mid \text{finite}\, g \land \text{finite}\, g' \land \#g = \#g' \vdash \\ \quad \exists\, \text{Isomorphism}[X, Y] \bullet a = g \land b = g'$$

Hence any finite field with a prime size p is isomorphic to the integers modulo p.

•

Ordered domain

Intent

An ordered integral domain is an integral domain with a total order defined on its elements.

Definition

OrderedIntegralDomain $[X]$

$\text{IntegralDomain}[X]$

$\_ \prec \_ : X \leftrightarrow X$

$\_ \preccurlyeq \_ : X \leftrightarrow X$

$g \lhd (\_ \prec \_) \rhd g \in \text{irreflexiveTotalOrder}\, g$

$g \lhd (\_ \preccurlyeq \_) \rhd g \in \text{reflexiveTotalOrder}\, g$

$g \lhd (\_ \preccurlyeq \_) \rhd g = (g \lhd (\_ \prec \_) \rhd g) \cup \text{id}\, g$

$O \prec I$

$\forall\, x, y, z : g \mid O \prec z \bullet x \prec y \Leftrightarrow x \dot{+} z \prec y \dot{+} z$

$\forall\, x, y, z : g \mid O \prec z \bullet x \prec y \Leftrightarrow x \dot{*} z \prec y \dot{*} z$

Examples

1. The integers under addition and multiplication form an ordered integral domain

$$\vdash \exists\, \text{OrderedIntegralDomain}[\mathbb{A}] \bullet \\ \quad g = \mathbb{N} \land (\_ \dot{+} \_) = (\_ + \_) \land O = 0 \land (\dot{-}\_) = (-\_) \\ \quad \land (\_ \dot{*} \_) = (\_ * \_) \land I = 1 \\ \quad \land (\_ \prec \_) = (\_ < \_)$$

Laws

■ **Law 28.7** A non-zero element and its additive inverse fall on either side of zero.

$$[X]\ \text{OrderedIntegralDomain}[X];\ x : X \mid O \neq x \in g \vdash O \prec x \Leftrightarrow \dot{-}\, x \prec O$$

■ **Law 28.8** A plain integral domain may be finite or infinite. Imposing a total order on its elements and requiring that addition of unity, for example, creates a greater element implies that an ordered integral domain must necessarily be infinite.

The elements of an ordered integral domain are unbounded above and below.

$$[X]\ \text{OrderedIntegralDomain}[X] \vdash \neg\ \text{finite}\, g$$
$$[X]\ \text{OrderedIntegralDomain}[X];\ x : X \mid x \in g \vdash \exists\, y, y' : g \bullet y \prec x \prec y'$$

—————————————•

Ordered field

Intent

An ordered field is a field with a total order.

Definition

$$\text{OrderedField}[X] == \text{Field}[X] \land \text{OrderedIntegralDomain}[X]$$

Examples

1. The rationals under addition and multiplication form an ordered field.

$$\begin{array}{l} \vdash \exists\, \text{OrderedField}[\mathbb{A}] \bullet \\ \quad g = \mathbb{Q} \\ \quad \land\ (\_ \dot{+} \_) = (\_ + \_) \land O = 0 \land (\dot{-}\, \_) = (-\_) \\ \quad \land\ (\_ \dot{*} \_) = (\_ * \_) \land I = 1 \land (\_^{\dot{-}1}) = (\_^{-1}) \\ \quad \land\ (\_ \prec \_) = (\_ < \_) \end{array}$$

Laws

■ **Law 28.9** An element and its multiplicative inverse are either both greater than zero, or both less than zero.

$$[X]\ \text{OrderedField}[X];\ x : X \mid O \neq x \in g \vdash O \prec x \Leftrightarrow O \prec x^{\dot{-}1}$$

■ **Law 28.10** As with integral domains, the inclusion of the total ordering means that, whereas fields may be finite, ordered fields are necessarily infinite. (A finite field is also called a *Galois field*, commemorating the contribution to group theory made by the French mathematician Evariste Galois, 1811–1832.)

The elements of an ordered field are unbounded above and below.

The elements of an ordered field are *dense*; between any two distinct elements there is always another.

$$[X]\ \text{OrderedField}[X] \vdash \neg\ \text{finite}\, g$$
$$[X]\ \text{OrderedField}[X];\ x : X \mid x \in g \vdash \exists\, y, y' : g \bullet y \prec x \prec y'$$
$$[X]\ \text{OrderedField}[X];\ y, y' : X \mid \{y, y'\} \subseteq g \land y \prec y' \vdash \exists\, x : g \bullet y \prec x \prec y'$$

•

Complete field

Intent

A complete field is an ordered field that contains its least upper bounds.

Definition

CompleteField $[X]$

$$\text{OrderedField}[X]$$

$$\forall\, a : \mathbb{P}_1\, g \mid \text{upperBound}((\_ \preccurlyeq \_) \rhd g)\, a \neq \varnothing \bullet a \in \text{dom}(\text{lub}((\_ \preccurlyeq \_) \rhd g))$$

Examples

1. The reals under addition and multiplication form a complete field.

$$\vdash \exists\, \mathrm{CompleteField}[\mathbb{A}] \bullet \\ g = \mathbb{R} \\ \land\ (\_ \dot{+} \_) = (\_ + \_) \land O = 0 \land (\dot{-}\_) = (-\_) \\ \land\ (\_ \dot{*} \_) = (\_ * \_) \land I = 1 \land (\_^{\dot{-}1}) = (\_^{-1}) \\ \land\ (\_ \prec \_) = (\_ < \_)$$

   The definition of *CompleteField* allows us to characterise the real numbers (§29). The rational numbers are a dense set, by virtue of being an ordered field. The ordering is not *complete*, however: there can be a number that we want to express that is between the rationals, but does not correspond to any one of them (for example, the square root of 2).

$$\vdash \exists\, a : \mathbb{P}\,\mathbb{Q} \mid a \in \mathrm{dom}(\mathrm{lub}((\_ < \_) \rhd \mathbb{Q})) \bullet \mathrm{lub}(\_ < \_)\, a \notin \mathbb{Q}$$

   The ordering of the real numbers has this property of completeness, which is defined by the guaranteed existence of the least upper bound for any non-empty bounded set. (We could equivalently work with the greatest lower bound.)

•

Part V

Numbers

Chapter 29

Axiomatic Properties of Numbers

Integers are widely used in Z specifications. Often quantities are declared to have integer type merely so their elements can be ordered or otherwise structured. The previous chapters have built up many structures on relations that can be used instead of resorting to a numeric type.

Being restricted to integers can sometimes be a limitation, however. Some problems are most naturally specified using real numbers. Clearly, if such a specification is later to be implemented, some numerical analysis will be needed. However, it is better to do that at a later refinement stage than build it into the specification.

In this chapter we provide support for the real numbers. We do this in an extensible manner, by considering the reals as embedded in some wider numeric set of the same type, not further constrained. An obvious such containing type, for example, is the complex numbers, and enhancement to support complex numbers would be straightforward; nothing is defined here that would need to be contradicted. Other sets that might be catered for are: an interpretation of numbers that allows infinite values; dimensioned numbers; intervals; vectors, quaternions, matrices, tensors; and so on.

The development here is axiomatic. We give a collection of properties, with the assumption that together they are satisfiable and that they characterise the real numbers as recognised by the mathematical community. We draw heavily on the properties built up in the chapter on groups, rings, and fields (§23).

- the numeric type arithmos: $\mathbb{A}$
- less than: $(\_ < \_)$, $(\_ \leq \_)$
- addition, multiplication: $(\_ + \_)$, $(\_ * \_)$
- negation: $(\text{-}\_)$
- integer properties, integers: $\mathbb{Z}$

- negative, positive, non-negative, non-zero numbers: $(\_\ \_)$, $(\_ +)$, $(\_ \oplus)$, $(\_ \pm)$
- natural numbers: $\mathbb{N}$
- reciprocal: $(\_^{-1})$
- rational properties, rational numbers: $\mathbb{Q}$
- real properties, real numbers: $\mathbb{R}$
- subtraction, division: $(\_ - \_)$, $(\_ \div \_)$
- other comparison relations: $(\_ \geq \_)$, $(\_ > \_)$

A construction of the real numbers from the bottom upwards, starting from Z free types, is given in [Valentine 1993b], [Valentine 1993a]. That development treats the reals as embedded in a set of intervals, whose properties could be added to the treatment given here, if desired.

For a readable technical account of the historical development of the concept of number, see [Ebbinghaus *et al.* 1991].

Concrete syntax for number literals

Intent

Provide a concrete syntax for writing number literals in Z.

Definitions

We affirm the *Standard* Z convention, which cannot be stated formally within the mechanism of Z but is assumed to be a part of the concrete syntax of Z, that any lexeme wholly composed of more than one occurrence of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, which we write as sd where s is a digit string and d is a digit, has the value given recursively by the rule

$$sd = (10 * s) + d$$

It is possible to write simple expressions to represent rational numbers, for example -3, $355 \div 113$

Standard Z does not define reals, so it need define no concrete syntax for them. We assume the traditional "decimal point" extension to the concrete syntax of number literals.

•

Standard Prelude

Intent

Define some elementary numerical properties required to support number literal expressions.

Definitions

The Z Standard prelude defines these properties in its "Standard Prelude". It declares a numeric type $\mathbb{A}$ (pronounced 'arithmos'), whose whole extent is deliberately not described, to allow extensions to be added.

$$[\mathbb{A}]$$

It declares the set of natural numbers, $\mathbb{N}$, to be a subset of $\mathbb{A}$

$$\mathbb{N} : \mathbb{P}\,\mathbb{A}$$

It declares a template for the addition operator.

$$\mathrm{function}\ 30\ \mathrm{leftassoc}\ (\_ + \_)$$

It declares addition over the entire set $\mathbb{A}$, and constrains it to have the expected properties when restricted to $\mathbb{N}$.

We start from these minimal definitions and extend them to further arithmetic operators, and larger subsets of $\mathbb{A}$.

Basic numerical operators

Intent

We declare, but give no further properties yet, the basic numeric comparison relation and the arithmetic operators for multiplication and negation.

Definitions

relation $(\_ < \_)$
relation $(\_ \leq \_)$
function 40 leftassoc $(\_ * \_)$
function $(\text{-}\_)$

The operator template for unary minus ensures that the expression $\text{--}x$ means $\text{-}(\text{-}x)$, as expected; if it were not defined using a templete it would mean $(\text{--})x$, not what is wanted.

$$\_ < \_ : \mathbb{A} \leftrightarrow \mathbb{A}$$
$$\_ \leq \_ : \mathbb{A} \leftrightarrow \mathbb{A}$$
$$\_ * \_ : \mathbb{A}^{2\times} \nrightarrow \mathbb{A}$$
$$\text{-}\_ : \mathbb{A} \nrightarrow \mathbb{A}$$

Integers and natural numbers

Intent

The integers form an ordered integral domain, with 'less than' as the total order. The additive identity element is zero; the multiplicative identity element is one. The set $\mathbb{Z}$ of integers is the smallest set with the integer properties.

Definitions

IntegerProperties

OrderedIntegralDomain[$\mathbb{A}$]

$$(\_ \dot{+} \_) = (\_ + \_)$$
$$O = 0$$
$$(\dot{\text{-}} \_) = (\text{-}\_)$$
$$(\_ \dot{*} \_) = (\_ * \_)$$
$$I = 1$$
$$(\_ \prec \_) = (\_ < \_)$$
$$(\_ \preccurlyeq \_) = (\_ \leq \_)$$

$$\mathbb{Z} : \mathbb{P}\,\mathbb{A}$$

$$\exists\,\mathrm{IntegerProperties} \bullet \mathbb{Z} = g$$
$$\forall\,\mathrm{IntegerProperties} \bullet \mathbb{Z} \subseteq g$$

This definition ensures that the set $\mathbb{Z}$ that we are defining is 'small enough' to be just the set of the integers; there are no spurious 'junk' elements present. Remember that the set g in an ordered integral domain is necessarily infinite, so this set as defined is also 'big enough' to hold all the integers.

We define operators to yield just negative numbers, just positive numbers, just non-negative numbers, and just non-zero numbers.

$$\mathrm{function}\,(\__-)$$
$$\mathrm{function}\,(\__+)$$
$$\mathrm{function}\,(\__\oplus)$$
$$\mathrm{function}\,(\__\pm)$$

$$\__-\ ,\ \__+\ ,\ \__\oplus\ ,\ \__\pm : \mathbb{P}\,\mathbb{A} \to \mathbb{P}\,\mathbb{A}$$

$$\forall\, a : \mathbb{P}\,\mathbb{A} \bullet$$
$$a_- = \{\ x : a \mid x < 0\ \}$$
$$\land\ a_+ = \{\ x : a \mid 0 < x\ \}$$
$$\land\ a_\oplus = \{0\} \cup a_+$$
$$\land\ a_\pm = a_- \cup a_+$$

The non-zero natural numbers are conventionally called $\mathbb{N}_1$.

$$\mathbb{N}_1 == \mathbb{N}_+$$

Laws

Law 29.1 The natural numbers are the non-negative subset of the integers.

$$\vdash \mathbb{N} = \mathbb{Z}_\oplus$$

Law 29.2 The non-zero natural numbers are the positive natural numbers.

$$\vdash \mathbb{N}_{\pm} = \mathbb{N}_{+}$$

The definitions allow us to deduce all the expected properties of $\mathbb{N}$.

Law 29.3 The set of natural numbers has the induction property: any set of numbers that includes zero and all its successors contains the natural numbers.

$$a : \mathbb{P}\,\mathbb{A} \mid 0 \in a \wedge (\ \forall\, n : a \bullet n + 1 \in a\ ) \vdash \mathbb{N} \subseteq a$$

Law 29.4 The set of natural numbers is unbounded above (there is always a greater natural number) and bounded below by zero.

$$n : \mathbb{N} \vdash \exists\, m : \mathbb{N} \bullet n < m$$
$$\vdash \neg\ \exists\, n : \mathbb{N} \bullet n < 0$$

Law 29.5 The set of integers is unbounded above and below. The set of integers is not *dense*.

$$i : \mathbb{Z} \vdash \exists\, j, k : \mathbb{Z} \bullet j < i < k$$
$$i : \mathbb{Z} \vdash \neg\ \exists j : \mathbb{Z} \bullet i < j < i + 1$$

■ **Law 29.6** There is a bijection between the integers and natural numbers.

$$\vdash \exists f : \mathbb{N} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{Z} \bullet \mathit{true}$$

———————————————•

Rational numbers

Intent

The rationals form an ordered field, with integer properties. The set $\mathbb{Q}$ of rationals is the smallest set with the rational properties.

Definitions

function $(\_^{-1})$

$$\_^{-1} : \mathbb{A} \pfun \mathbb{A}$$

RationalProperties

OrderedField[$\mathbb{A}$]
IntegerProperties

$$(\_^{\dot{-}1}) = (\_^{-1})$$

$$\mathbb{Q} : \mathbb{P}\,\mathbb{A}$$

$$\exists\, \text{RationalProperties} \bullet \mathbb{Q} = g$$
$$\forall\, \text{RationalProperties} \bullet \mathbb{Q} \subseteq g$$

Note that the set $\mathbb{Q}_{\pm}$ corresponds to the set g_0 in the *OrderedField* part of *RationalProperties*.

Laws

Law 29.7 The non-zero rationals form an abelian group under multiplication.

$$\vdash \exists\, \text{AbelianGroup}[\mathbb{A}] \bullet \\ g = \mathbb{Q}_{\pm} \land (\_ \diamond \_) = (\_ * \_) \land e = 1 \land inv = (\_^{-1})$$

Law 29.8 The rationals are dense: for any two distinct rationals, there is another rational that lies between them.

$$p, p' : \mathbb{Q} \mid p < p' \vdash \exists\, q : \mathbb{Q} \bullet p < q < p'$$

■ **Law 29.9** There is a bijection between the natural numbers and pairs of naturals. There is a bijection between the natural numbers and the rationals.

$$\vdash \exists f : \mathbb{N} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{N}^{2\times} \bullet \mathit{true}$$
$$\vdash \exists f : \mathbb{N} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{Q} \bullet \mathit{true}$$

•

Real numbers

Intent

The reals form a complete field. The set $\mathbb{R}$ of reals is the smallest set with the real properties.

Definitions

RealProperties

CompleteField[$\mathbb{A}$]
RationalProperties

$$\mathbb{R} : \mathbb{P}\,\mathbb{A}$$

$$\exists\, \text{RealProperties} \bullet \mathbb{R} = g$$
$$\forall\, \text{RealProperties} \bullet \mathbb{R} \subseteq g$$

Examples

1. The square root of two is a real, but not a rational, number.
   $\sqrt{2} = \text{lub}((\_ < \_) \cap \mathbb{R}^{2\times})\{\ x : \mathbb{Q} \mid x * x \leq 2\ \}$

Laws

Law 29.10 The non-zero reals form an abelian group under multiplication.

$$\vdash \exists\, \text{AbelianGroup}[\mathbb{A}] \bullet \\ g = \mathbb{R}_{\pm} \land (\_ \diamond \_) = (\_ * \_) \land e = 1 \land inv = (\_^{-1})$$

Law 29.11 The rationals are dense in $\mathbb{R}$: for any two distinct reals, there is a rational that lies between them.

$$x, y : \mathbb{R} \mid x < y \vdash \exists\, q : \mathbb{Q} \bullet x < q < y$$

■ **Law 29.12** There is a bijection between the power set of natural numbers and the reals.

$$\vdash \exists f : \mathbb{P}\,\mathbb{N} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{R} \bullet \mathit{true}$$

Law 29.13 As a corollary of law 29.12, with law 21.26, there is no bijection between the naturals and the reals.

$$\vdash \mathbb{N} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{R} = \varnothing$$

Subtraction and division

Intent

We give a declaration for these operators wide enough to allow the operation on all $\mathbb{A}$s, but *define* them only for reals, leaving them loose outside this set, allowing different extensions as appropriate.

Definitions

function 30 leftassoc $(\_ - \_)$

function 40 leftassoc $(\_ \div \_)$

$$\begin{array}{|l} \_ - \_ : \mathbb{A}^{2\times} \nrightarrow \mathbb{A} \\ \hline (\ \lambda\, x, y : \mathbb{R} \bullet x + \text{-}y\ ) \subseteq (\_ - \_) \end{array}$$

$$\begin{array}{|l} \_ \div \_ : \mathbb{A}^{2\times} \nrightarrow \mathbb{A} \\ \hline \{\ x, y : \mathbb{R};\ z : \mathbb{R}_{\pm} \mid x = y * z \bullet (x, z) \mapsto y\ \} \subseteq (\_ \div \_) \end{array}$$

Laws

Law 29.14 Subtraction from zero, and division into one, are the additive and multiplicative inverse functions.

$$\begin{array}{l} \mathrm{RealProperties} \mid g = \mathbb{R} \vdash \\ \quad \mathbb{R} \lhd (\text{-}\_) = (\ \lambda\, x : \mathbb{R} \bullet 0 - x\ ) \\ \quad \land\ \mathbb{R} \lhd (\_^{-1}) = (\ \lambda\, x : \mathbb{R}_{\pm} \bullet 1 \div x\ ) \end{array}$$

Numerical orders

Intent

We give a declaration for numerical orders wide enough to allow an order on all $\mathbb{A}$s, but *define* them only for reals, leaving them loose outside this set, allowing different extensions as appropriate.

Definitions

relation $(\_ \geq \_)$

relation $(\_ > \_)$

$$\_ \geq \_, \_ > \_ : \mathbb{A} \leftrightarrow \mathbb{A}$$

$$\forall\, x, y : \mathbb{R} \bullet x > y \Leftrightarrow y < x$$

$$\forall\, x, y : \mathbb{R} \bullet x \geq y \Leftrightarrow y \leq x$$

•

Chapter 30

Further Numbers

In the previous chapter we introduced the minimum of definitions needed to define the real numbers. In this chapter we introduce further operations applicable to numbers.

- sign, absolute value: sign, abs
- floor, ceiling: $\lfloor \_ \rfloor$, $\lceil \_ \rceil$
- integer division, modulus: div, mod
- integer range: $(\_ \mathinner{..} \_)$
- cardinality, total cardinality: $\#$, $\vec{\#}$
- maximum, minimum (of a set of numbers): max, min
- composite numbers, primes, coprimes
- square root: relational $\sqrt{}$, and functional $\sqrt{\ }$

Sign

Intent

The *sign* function gives -1, 1 or 0, depending on whether its argument is negative, positive, or zero, respectively.

Definition

function $(\text{sign}\ \_)$

$$\text{sign}\ \_ : \mathbb{A} \nrightarrow \mathbb{A}$$

$$(\ \lambda\, x : \mathbb{R} \bullet \text{-1}\ ) \oplus (\ \lambda\, x : \mathbb{R}_+ \bullet 1\ ) \oplus \{0 \mapsto 0\} \subseteq (\text{sign}\ \_)$$

If there is any danger of confusion in the pronunciation of *sign* with *sine*, it is usual to pronounce the former as the alternative Latin form *signum*.

Laws

Law 30.1 The *sign* function is total on the reals; its range is $\{\text{-}1, 0, 1\}$.

$$\vdash \mathbb{R} \lhd (\text{sign}\_) \in \mathbb{R} \twoheadrightarrow \{\text{-}1, 0, 1\}$$

Absolute value

Intent

The *abs* function gives the positive form of its argument.

Definition

function $(\text{abs}\_)$

$$\begin{array}{|l} \text{abs}\_ : \mathbb{A} \nrightarrow \mathbb{A} \\ \hline \text{id}\,\mathbb{R} \oplus (\ \lambda\, x : \mathbb{R}\_ \bullet \text{-}x\ ) \subseteq (\text{abs}\_) \end{array}$$

Laws

Law 30.2 The *abs* function is total on the reals; its range is the non-negative reals.

$$\vdash \mathbb{R} \lhd (\text{abs}\_) \in \mathbb{R} \twoheadrightarrow \mathbb{R}_{\oplus}$$

Law 30.3 The *abs* function is idempotent.

$$x : \mathbb{R} \vdash \text{abs}(\text{abs}\, x) = \text{abs}\, x$$

Floor and ceiling

Intent

Floor, or 'integer part', rounds a real number x down to the nearest integer below. Ceiling rounds a real number x up to the nearest integer above.

Definition

floor

function $(\lfloor \_ \rfloor)$

$$\frac{\lfloor \_ \rfloor : \mathbb{A} \pfun \mathbb{A}}{\{\ x : \mathbb{R};\ i : \mathbb{Z} \mid 0 \leq x - i < 1\ \} \subseteq (\lfloor \_ \rfloor)}$$

ceiling

function $(\lceil \_ \rceil)$

$$\frac{\lceil \_ \rceil : \mathbb{A} \pfun \mathbb{A}}{\{\ x : \mathbb{R};\ i : \mathbb{Z} \mid 0 \leq i - x < 1\ \} \subseteq (\lceil \_ \rceil)}$$

Laws

Law 30.4 The floor and ceiling functions are total on the reals; their ranges are the integers.

$$\vdash \mathbb{R} \lhd (\lfloor \_ \rfloor) \in \mathbb{R} \tfun \mathbb{Z}$$
$$\vdash \mathbb{R} \lhd (\lceil \_ \rceil) \in \mathbb{R} \tfun \mathbb{Z}$$

Law 30.5 A real number lies between the floor and the ceiling.

$$x : \mathbb{R} \vdash x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$$

Law 30.6 The floor and ceiling on integers is the identity; the floor and ceiling of non-integers differ by unity.

$i : \mathbb{Z} \vdash \lfloor i \rfloor = \lceil i \rceil = i$
$x : \mathbb{R} \setminus \mathbb{Z} \vdash \lfloor x \rfloor + 1 = \lceil x \rceil \neq x$

———————————•

Integer division and modulus

Intent

div gives the integer part of the quotient (real division) rounded down (towards minus infinity). The function *div* is here defined for real arguments; the word 'integer' in the name refers to the fact that its *result* is an integer.

mod gives the remainder after integer division.

Definition

integer division

function 40 leftassoc (_ div _)

$$\begin{array}{|l} \_ \operatorname{div} \_ : \mathbb{A}^{2\times} \nrightarrow \mathbb{A} \\ \hline (\ \lambda\, x : \mathbb{R};\ y : \mathbb{R}_{\pm} \bullet \lfloor x \div y \rfloor\ ) \subseteq (\_ \operatorname{div} \_) \end{array}$$

modulus

function 40 leftassoc (_ mod _)

$$\begin{array}{|l} \_ \operatorname{mod} \_ : \mathbb{A}^{2\times} \nrightarrow \mathbb{A} \\ \hline (\ \lambda\, x : \mathbb{R};\ y : \mathbb{R}_{\pm} \bullet x - (x \operatorname{div} y) * y\ ) \subseteq (\_ \operatorname{mod} \_) \end{array}$$

Examples

1. 4.2 div 3 = 1; 4.2 mod 3 = 1.2
2. -4.2 div 3 = -2; -4.2 mod 3 = 1.8

3. 4.2 div -3 = -2; 4.2 mod -3 = -1.8
4. -4.2 div -3 = 1; -4.2 mod -3 = -1.2

The decision of how to treat negative divisors is contentious; every conceivable policy, and some inconceivable ones, are currently on offer from some author or system. We have chosen to follow the Standard Z Mathematical Toolkit definitions. The major debate involves whether to round the result downwards, or truncate to zero (these having the same effect for positive divisors).

Building a new definition to treat negative divisors differently is completely straightforward. It would be sensible for any such redefinition to follow the standard definition for positive divisors. In practice negative divisors are never used.

> *"What, never?"*
> *"No, never!"*
> *"What, never?"*
> *"Hardly ever!"*
>
> — W. S. Gilbert, *H.M.S. Pinafore*, 1878

Laws

Law 30.7 The *div* function is total on pairs comprising a real and a non-zero real; its range is the integers. The *mod* function is total on pairs comprising a real and a non-zero real; its range is the reals.

$$\vdash (\mathbb{R} \times \mathbb{R}_{\pm}) \lhd (\_ \operatorname{div} \_) \in (\mathbb{R} \times \mathbb{R}_{\pm}) \pfun \mathbb{Z}$$
$$\vdash (\mathbb{R} \times \mathbb{R}_{\pm}) \lhd (\_ \bmod \_) \in (\mathbb{R} \times \mathbb{R}_{\pm}) \pfun \mathbb{R}$$

Law 30.8 The remainder when dividing by a positive number is also positive, and less than that number.

$$x : \mathbb{R};\ y : \mathbb{R}_+ \vdash 0 \leq x \bmod y < y$$

Law 30.9 y divides x precisely when there is no remainder.

$$x : \mathbb{R};\ y : \mathbb{R}_{\pm} \vdash x \bmod y = 0 \Leftrightarrow (\ \exists\, i : \mathbb{Z} \bullet x = i * y\ )$$

Law 30.10 Any real number can be broken into its integer part and remainder.

$$x : \mathbb{R} \vdash x = \lfloor x \rfloor + (x \bmod 1)$$

Law 30.11 A non-zero number can be cancelled from the 'top' and 'bottom' of an integer division.

$$x : \mathbb{R};\ y, z : \mathbb{R}_{\pm} \vdash (x * z) \operatorname{div} (y * z) = x \operatorname{div} y$$

Law 30.12 The modulus operation forms a homomorphism from addition to addition modulo y, and from multiplication to multiplication modulo y.

$$x, x' : \mathbb{R};\ y : \mathbb{R}_{\pm} \vdash ((x \bmod y) + (x' \bmod y)) \bmod y = (x + x') \bmod y$$
$$x, x' : \mathbb{R};\ y : \mathbb{R}_{\pm} \vdash ((x \bmod y) * (x' \bmod y)) \bmod y = (x * x') \bmod y$$

•

Integer range

Intent

The function $\_ \mathinner{..} \_$ gives the set of all integers between a beginning and an ending number, which themselves need not be integer.

Definition

function 20 leftassoc $(\_ \mathinner{..} \_)$

$$\_ \mathinner{..} \_ : \mathbb{A}^{2\times} \nrightarrow \mathbb{P}\,\mathbb{Z}$$
$$(\ \lambda\, x, y : \mathbb{R} \bullet \{\ i : \mathbb{Z} \mid x \le i \le y\ \}\ ) \subseteq (\_ \mathinner{..} \_)$$

Examples

1. $3 \mathinner{..} 6 = \{3, 4, 5, 6\}$
2. $\pi \mathinner{..} 6 = \{4, 5, 6\}$

Laws

Law 30.13 If the beginning number is greater than the ending number, then the integer range is empty.

$$x, y : \mathbb{R} \mid y < x \vdash x \mathinner{..} y = \varnothing$$

Law 30.14 If the beginning number is an integer, and equal to the ending number, then the integer range is the singleton set containing that number.

$$i : \mathbb{Z} \vdash i \mathinner{..} i = \{i\}$$

Law 30.15 If the beginning number is not an integer, and equal to the ending number, then the integer range is empty.

$$x : \mathbb{R} \setminus \mathbb{Z} \vdash x \mathinner{..} x = \varnothing$$

Law 30.16 An integer range is a subset of another if its beginning and ending numbers form a subinterval of the other's corresponding interval.

$$x, x', y, y' : \mathbb{R} \mid x' \leq x \land y \leq y' \vdash x \mathinner{..} y \subseteq x' \mathinner{..} y'$$

Law 30.17 The intersection of two integer ranges is an integer range. The union of two overlapping integer ranges is an integer range.

$$x, x', y, y' : \mathbb{R} \vdash (x \mathinner{..} y) \cap (x' \mathinner{..} y') = \max\{x, x'\} \mathinner{..} \min\{y, y'\}$$
$$x, x', y, y' : \mathbb{R} \mid x \leq x' \leq y \vdash (x \mathinner{..} y) \cup (x' \mathinner{..} y') = x \mathinner{..} y'$$

Law 30.18 The integer range is unaffected by rounding up the beginning number, and rounding down the ending number.

$$x, y : \mathbb{R} \vdash x \mathinner{..} y = \lceil x \rceil \mathinner{..} \lfloor y \rfloor$$

Law 30.19 Consider two integer ranges. New integer ranges can be made by taking the sum of all pairs from the two ranges, and the difference of all pairs.

$$\begin{aligned} &x, x', y, y' : \mathbb{Z} \mid x \leq y \land x' \leq y' \vdash \\ &\quad (\_ + \_)(\!| \, (x \mathinner{..} y, x' \mathinner{..} y') \, |\!) = (x + x') \mathinner{..} (y + y') \\ &\quad \land (\_ - \_)(\!| \, (x \mathinner{..} y, x' \mathinner{..} y') \, |\!) = (x - y') \mathinner{..} (y - x') \end{aligned}$$

Cardinality

Intent

The cardinality of a set is its size, the number of elements it contains.

Definition

The cardinality of a finite set is defined in terms of a bijection with an initial segment of the natural numbers.

$$\text{function}\,(\#\ \_)$$

The template for cardinality ensures that an expression like $\#\ \bigcup \alpha$ usefully means $\#(\bigcup \alpha)$; if it were a simple function it would mean $(\#\ \bigcup)\alpha$, not what is wanted.

$$[X]$$
$$\#\ \_ : \mathbb{P}\, X \pfun \mathbb{A}$$
$$\{\ a : \mathbb{P}\, X;\ n : \mathbb{N} \mid 1 \upto n \rightarrowtail\!\!\!\!\!\!\rightarrow\ a \neq \varnothing\ \} \subseteq (\#\ \_)$$

The definition is loose in that it says nothing about the result of applying the function # to an infinite set. A specifier could tighten this definition by establishing one or more elements of $\mathbb{A}$ to be the possible result of applying # to an infinite set.

Examples

1. $\#\ \varnothing = 0$
2. $\#\{x\} = 1$

Laws

Law 30.20 Any finite set has a cardinality.

$$[X] \vdash \mathbb{F}\, X \lhd (\#\ \_) \in \mathbb{F}\, X \fun \mathbb{N}$$

Law 30.21 The sizes of combinations of sets are related to the sizes of other combinations.

$$[X]\ a : \mathbb{F}\, X;\ b : \mathbb{P}\, X \vdash \#a = \#(a \setminus b) + \#(a \cap b)$$
$$[X]\ a, b : \mathbb{F}\, X \vdash \#a + \#b = \#(a \cup b) + \#(a \cap b)$$
$$[X]\ a, b : \mathbb{F}\, X \vdash \#(a \cup b) = \#a + \#(b \setminus a)$$
$$[X]\ a, b : \mathbb{F}\, X \vdash \#(a \cup b) = \#(a \ominus b) + \#(a \cap b)$$
$$[X]\ a, b : \mathbb{F}\, X \vdash \#(a \ominus b) = \#(a \setminus b) + \#(b \setminus a)$$

Law 30.22 Cardinality is order-preserving (§26).

$$[X]\ a, b : \mathbb{F}\, X \mid a \subseteq b \vdash \#a \leq \#b$$

Law 30.23 Specialising the order-preserving law about bounds (law 26.36) to $f = \#$ gives

$$[X]\ \alpha : \mathbb{F}(\mathbb{F}\, X) \vdash \max(\#(\!|\ \alpha\ |\!)) \leq \#(\bigcup \alpha)$$
$$[X]\ \alpha : \mathbb{F}(\mathbb{F}\, X) \vdash \#(\bigcap \alpha) \leq \min(\#(\!|\ \alpha\ |\!))$$
$$[X]\ a, b : \mathbb{F}\, X \vdash \max\{\#a, \#b\} \leq \#(a \cup b)$$
$$[X]\ a, b : \mathbb{F}\, X \vdash \#(a \cap b) \leq \min\{\#a, \#b\}$$

Law 30.24 Two finite sets have the same size precisely when there is a bijection between them. (This condition can be taken as a definition of 'same size' for infinite sets.)

$$[X, Y]\ a : \mathbb{F}\, X;\ b : \mathbb{F}\, Y \vdash \#a = \#b \Leftrightarrow (\ \exists f : a \rightarrowtail\!\!\!\!\!\rightarrow b \bullet \mathit{true}\ )$$

Law 30.25 The size of the domain of a relation (and dually, the size of its range) cannot be larger than the size of the relation. The size of a relation cannot be greater than the size of the cartesian product of its domain and range.

$$[X, Y]\ r : X \leftrightarrow\!\!\!\!\!\rightarrow Y \vdash \#(\mathrm{dom}\, r) \leq \#r \leq \#(\mathrm{dom}\, r) * \#(\mathrm{ran}\, r)$$

Law 30.26 A finite relation is functional precisely when its size is the same as the size of its domain. Dually, a finite function is injective precisely when its size is same as the size of its range. (Hence a finite injection has the same size domain and range.)

$$[X, Y]\ r : X \nrightarrow Y \vdash \#r = \#(\operatorname{dom} r) \Leftrightarrow r \in X \nrightarrow Y$$
$$[X, Y]\ f : X \nrightarrow Y \vdash \#f = \#(\operatorname{ran} f) \Leftrightarrow f \in X \nrightarrow Y$$

Law 30.27 The size of an integer range is given by its end points.

$$i, j : \mathbb{Z} \vdash \#(i \mathinner{..} j) = \max\{j - i + 1, 0\}$$

•

Total cardinality

Intent

The function $\vec{\#}$ is provided for use where there is a need to compare cardinalities of sets, some of which may be infinite.

Definition

generic $(\vec{\#}\_)$

$$\vec{\#} X == \lambda\, a : \mathbb{P}\, X \bullet \{\ b : \mathbb{F}\, a \bullet \#b\ \}$$

Examples

1. $n \in \vec{\#}\, a$ means that a has at least n elements
2. $\#\{a, b, c\} = 3 \vdash \vec{\#}\{a, b, c\} = \{0, 1, 2, 3\}$
3. $\vec{\#}\, \mathbb{N} = \mathbb{N}$
4. $\vec{\#}\, \mathbb{R} = \mathbb{N}$

Laws

Law 30.28 $\vec{\#}$ is a total function from sets to non-empty sets of natural numbers

$$[X]\ \vdash \vec{\#}\, X \in \mathbb{P}\, X \to \mathbb{P}_1\, \mathbb{N}$$

Law 30.29 For finite sets, the set $\vec{\#}\, X$ contains precisely the natural numbers from zero to the size of X.

$$[X] \ \vdash \text{finite}\, X \Rightarrow \vec{\#}\, X = 0 \mathinner{..} \#X$$

Law 30.30 A set is not finite precisely when its total cardinality is the set of all natural numbers.

$$[X] \ \vdash \neg\ \text{finite}\, X \Leftrightarrow \vec{\#}\, X = \mathbb{N}$$

Law 30.31 The total cardinality of set a is a subset of that of set b precisely when either b is not finite, or both sets are finite and a is not bigger than b.

$$[X] \ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash \vec{\#}\, a \subseteq \vec{\#}\, b \Leftrightarrow (\text{finite}\, b \Rightarrow \text{finite}\, a \wedge \#a \leq \#b)$$

Law 30.32 The total cardinality of set a is equal to that of set b precisely when either both sets are not finite, or both sets are finite and the same size.

$$[X] \ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \vdash \\ \vec{\#}\, a = \vec{\#}\, b \Leftrightarrow ((\text{finite}\, a \Leftrightarrow \text{finite}\, b) \wedge (\text{finite}\, a \Rightarrow \#a = \#b))$$

Law 30.33 There are total cardinality analogues of law 30.33.

$$[X] \ a, b : \mathbb{P}\, X \vdash \vec{\#}\, a = (\_ + \_)(\!|\ (\vec{\#}(a \setminus b), \vec{\#}(a \cap b))\ |\!)$$
$$[X] \ a, b : \mathbb{P}\, X \vdash (\_ + \_)(\!|\ (\vec{\#}\, a, \vec{\#}\, b)\ |\!) = (\_ + \_)(\!|\ (\vec{\#}(a \cup b), \vec{\#}(a \cap b))\ |\!)$$
$$[X] \ a, b : \mathbb{P}\, X \vdash \vec{\#}(a \cup b) = (\_ + \_)(\!|\ (\vec{\#}\, a, \vec{\#}(b \setminus a))\ |\!)$$
$$[X] \ a, b : \mathbb{P}\, X \vdash \vec{\#}(a \cup b) = (\_ + \_)(\!|\ (\vec{\#}(a \ominus b), \vec{\#}(a \cap b))\ |\!)$$
$$[X] \ a, b : \mathbb{P}\, X \vdash \vec{\#}(a \ominus b) = (\_ + \_)(\!|\ (\vec{\#}(a \setminus b), \vec{\#}(b \setminus a))\ |\!)$$

Law 30.34 Where there is a bijection between two sets, their total cardinality sets are equal.

$$[X] \ a : \mathbb{P}\, X;\ b : \mathbb{P}\, Y \mid a \rightarrowtail\!\!\!\!\!\rightarrow\ b \neq \varnothing \vdash \vec{\#}\, a = \vec{\#}\, b$$

Minimum and maximum

Intent

The minimum and maximum of a set of numbers are defined as special cases of the minimum and maximum functions with respect to the 'less than or equal to' reflexive ordering on real numbers.

Definition

$$\text{function}\,(\min\_)$$
$$\text{function}\,(\max\_)$$

The operator templates for min and max ensure that an expression like $\min \operatorname{ran} f$ usefully means $\min(\operatorname{ran} f)$; if it were a simple function it would mean $(\min \operatorname{ran}) f$, not what is wanted.

$$\begin{array}{|l} \min\_ : \mathbb{P}_1\, \mathbb{A} \pfun \mathbb{A} \\ \max\_ : \mathbb{P}_1\, \mathbb{A} \pfun \mathbb{A} \\ \hline \text{minimum}(\ (\_ \leq \_) \cap \mathbb{R}^{2\times}\ ) \subseteq (\min\_) \\ \text{maximum}(\ (\_ \leq \_) \cap \mathbb{R}^{2\times}\ ) \subseteq (\max\_) \end{array}$$

Laws

Law 30.35 Any non-empty set of natural numbers has a minimum element; any finite non-empty set of real numbers has a minimum and a maximum element.

$$\vdash \mathbb{P}_1\, \mathbb{N} \lhd (\min\_) \in \mathbb{P}_1\, \mathbb{N} \pfun \mathbb{N}$$
$$\vdash \mathbb{F}_1\, \mathbb{R} \lhd (\min\_) \in \mathbb{F}_1\, \mathbb{R} \pfun \mathbb{R}$$
$$\vdash \mathbb{F}_1\, \mathbb{R} \lhd (\max\_) \in \mathbb{F}_1\, \mathbb{R} \pfun \mathbb{R}$$

Law 30.36 Minimum is order-reversing (§26).

$$a, b : \operatorname{dom}(\min\_) \mid a \subseteq b \vdash \min b \leq \min a$$

Law 30.37 Specialising the order-reversing law about bounds (law 26.39) to $f = \min$ gives

$$\alpha : \mathbb{P}\,\mathbb{P}\,\mathbb{A} \mid \alpha \cup \{\bigcap \alpha\} \subseteq \mathrm{dom}(\min\_) \vdash \max(\min(\!|\ \alpha\ |\!)) \leq \min(\bigcap \alpha)$$
$$\alpha : \mathbb{P}\,\mathbb{P}\,\mathbb{A} \mid \alpha \cup \{\bigcup \alpha\} \subseteq \mathrm{dom}(\min\_) \vdash \min(\bigcup \alpha) = \min(\min(\!|\ \alpha\ |\!))$$
$$a, b : \mathrm{dom}(\min\_) \mid a \cap b \in \mathrm{dom}(\min\_) \vdash \max\{\min a, \min b\} \leq \min(a \cap b)$$
$$a, b : \mathrm{dom}(\min\_) \mid a \cup b \in \mathrm{dom}(\min\_) \vdash \min(a \cup b) = \min\{\min a, \min b\}$$

Law 30.38 Maximum is order-preserving (§26).

$$a, b : \mathrm{dom}(\max\_) \mid a \subseteq b \vdash \max a \leq \max b$$

Law 30.39 Specialising the order-preserving law about bounds (law 26.36) to $f = \max$ gives

$$\alpha : \mathbb{P}\,\mathbb{P}\,\mathbb{A} \mid \alpha \cup \{\bigcap \alpha\} \subseteq \mathrm{dom}(\max\_) \vdash \max(\max(\!|\ \alpha\ |\!)) = \max(\bigcap \alpha)$$
$$\alpha : \mathbb{P}\,\mathbb{P}\,\mathbb{A} \mid \alpha \cup \{\bigcup \alpha\} \subseteq \mathrm{dom}(\max\_) \vdash \max(\bigcap \alpha) \leq \min(\max(\!|\ \alpha\ |\!))$$
$$a, b : \mathrm{dom}(\max\_) \mid a \cup b \in \mathrm{dom}(\max\_) \vdash \max\{\max a, \max b\} = \max(a \cup b)$$
$$a, b : \mathrm{dom}(\max\_) \mid a \cap b \in \mathrm{dom}(\max\_) \vdash \max(a \cap b) \leq \min\{\max a, \max b\}$$

———————————•

Prime numbers

Intent

To define various prime and composite subsets of the natural numbers.

Definition

Composite numbers can be factored.

$$\mathbb{N}_2 == \mathbb{N} \setminus \{0, 1\}$$
$$\mathrm{composite} == \{\ n, m : \mathbb{N}_2 \bullet n * m\ \}$$

Prime numbers have no factors.

$$\mathrm{prime} == \mathbb{N}_2 \setminus \mathrm{composite}$$

Coprimes have no common factors.

$$\mathrm{relation}\,(\mathrm{coprime}\_)$$
$$\mathrm{coprime}\_ == \mathbb{N}_2^{2\times} \setminus \{\ n, m : \mathbb{N}_1;\ i : \mathbb{N}_2 \bullet (n * i, m * i)\ \}$$

Examples

1. composite $= \{4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, \ldots\}$
2. prime $= \{2, 3, 5, 7, 11, 13, 17, 19, \ldots\}$
3. coprime$(2, 3)$; coprime$(4, 9)$; coprime$(15, 14)$

Laws

Law 30.40 The set of primes is infinite.

$$\vdash \neg \text{ finite } \text{ prime}$$

Law 30.41 *Fundamental Theorem of Arithmetic*: any positive whole number can be uniquely factored into primes.

$$n : \mathbb{N}_2 \vdash \exists_1 f : \text{prime} \nrightarrow \mathbb{N}_1 \bullet n = \Pi \ (\ \lambda p : \text{dom}\, f \bullet p \ast\ast (f\ p)\ )$$

■ **Law 30.42** p is prime precisely when p divides $(p-1)! + 1$

$$p : \mathbb{N}_2 \vdash p \in \text{prime} \Leftrightarrow (\text{factorial}(p-1) + 1) \bmod p = 0$$

Law 30.43 Two numbers are coprime when they have no factors in common.

$$n, m : \mathbb{N}_2 \vdash \text{coprime}(n, m) \Leftrightarrow \text{disjoint}\langle \text{dom}(\text{factor}\, n), \text{dom}(\text{factor}\, m)\rangle$$

•

Square root

Intent

Two square root functions are provided. One is a relation between numbers and their (positive and negative) roots; one is a function from a number to its positive root.

Definition

$$\begin{array}{|l} \sqrt{} : \mathbb{A} \leftrightarrow \mathbb{A} \\ \hline \{\ x, y : \mathbb{R} \mid (x, x) \in \mathrm{dom}(\_ * \_) \wedge x * x = y\ \} \subseteq \sqrt{} \end{array}$$

function $(\sqrt{\_})$

$\sqrt{\_} == \sqrt{} \cap (\mathbb{A} \times \mathbb{R}_{\oplus})$

Examples

1. $\sqrt{}(\!|\ \{4\}\ |\!) = \{-2, 2\}$
2. $\sqrt{4} = 2$

Laws

Law 30.44 The square root is the one half power.

$x : \mathbb{R} \vdash \sqrt{x} = x **(1 \div 2)$

Chapter 31

Numbers and Relations

Now that we have defined numbers, we can define numeric properties of relations, such as iterating a homogeneous relation a certain number of times, and we can define the cardinality of certain properties of relations, such as the number of arcs leaving a vertex.

- relational iteration: $(\_^{-})$
- vertex degree: outDegree, inDegree, degree

Relation iteration

Intent

Relational iteration is composition of a relation with itself, n times.

A specification that makes heavy use of a particular number of iterations may well be over-specific. It may be possible to generalise it to use 'iteration transitive closure', r^+ or r^* (§24), or 'maximal iteration' (§24).

Definition

function $(\_^{-})$

$$
\begin{aligned}
\_^{-}\,[X] == \bigcap \{\; & f : (X \leftrightarrow X) \times \mathbb{N} \leftrightarrow (X \leftrightarrow X) \mid \\
& (\ \forall r : X \leftrightarrow X \bullet ((r, 0), \mathrm{id}\, X) \in f\ ) \\
& \land\ (\ \forall n : \mathbb{N};\ r, s : X \leftrightarrow X \mid ((r, n), s) \in f \bullet \\
& \qquad ((r, n+1), r \fatsemi s) \in f\ )\ \}
\end{aligned}
$$

Laws

Law 31.1 Relational iteration is a total surjection. (It is a surjection because of the identity $r^1 = r$.)

$$[X] \ \vdash (\_^{-})\ [X] \in (X \leftrightarrow X) \times \mathbb{N} \twoheadrightarrow (X \leftrightarrow X)$$

Law 31.2 An equivalent expression for relational iteration, explicitly in terms of composition, is

$$[X]\ r : X \leftrightarrow X \vdash r^0 = \mathrm{id}\, X$$
$$[X]\ r : X \leftrightarrow X;\ n : \mathbb{N} \vdash r^{n+1} = r \fatsemi r^n = r^n \fatsemi r$$

Law 31.3 Iteration commutes with inverse

$$[X]\ r : X \leftrightarrow X;\ n : \mathbb{N} \vdash r^{\sim n} = r^{n\sim}$$

Law 31.4 Composing iterations is equivalent to iterating by the sum. Iterating iterations is equivalent to iterating by the product.

$$[X]\ r : X \leftrightarrow X;\ n, m : \mathbb{N} \vdash r^n \fatsemi r^m = r^{n+m}$$
$$[X]\ r : X \leftrightarrow X;\ n, m : \mathbb{N} \vdash (r^n)^m = r^{n*m}$$

■ **Law 31.5** Iteration is subset order-preserving (§26) on its relation argument.

$$[X]\ r, s : X \leftrightarrow X;\ n : \mathbb{N} \mid r \subseteq s \vdash r^n \subseteq s^n$$

Law 31.6 Specialising the order-preserving laws about bounds (laws 26.36) to $f = \_^n$ gives

$$[X]\ \rho : \mathbb{P}(X \leftrightarrow X);\ n : \mathbb{N} \vdash \bigcup\{\ r : \rho \bullet r^n\ \} \subseteq (\bigcup \rho)^n$$
$$[X]\ \rho : \mathbb{P}(X \leftrightarrow X);\ n : \mathbb{N} \vdash (\bigcap \rho)^n \subseteq \bigcap\{\ r : \rho \bullet r^n\ \}$$
$$[X]\ r, s : X \leftrightarrow X;\ n : \mathbb{N} \vdash r^n \cup s^n \subseteq (r \cup s)^n$$
$$[X]\ r, s : X \leftrightarrow X;\ n : \mathbb{N} \vdash (r \cap s)^n \subseteq r^n \cap s^n$$

■ **Law 31.7** If the composition of two relations is commutative, then so is an iteration of one composed with another iteration of the other.

$$[X]\ r, s : X \leftrightarrow X;\ n, m : \mathbb{N} \mid r \fatsemi s = s \fatsemi r \vdash r^n \fatsemi s^m = s^m \fatsemi r^m$$

■ **Law 31.8** If the composition of two relations is commutative, then the iteration of their composition is the same as the composition of their iterations.

$$[X]\ r, s : X \leftrightarrow X;\ n : \mathbb{Z} \mid r \fatsemi s = s \fatsemi r \vdash (r \fatsemi s)^n = r^n \fatsemi s^n$$

———————————•

Vertex degree

Intent

Here we develop some more theory of homogeneous binary relations (§24), in terms of their *degree*, the number of arcs entering or leaving a vertex.

Definition

out degree of a vertex:

The *outDegree* of a vertex is the number of arcs leaving that vertex.

$$\begin{aligned}&\mathrm{outDegree}[X] == \\ &\quad \lambda\, r : X \leftrightarrow X \bullet \lambda\, x : X \mid \mathrm{finite}(\mathrm{successors}\ r\ x) \bullet \#\,\mathrm{successors}\ r\ x\end{aligned}$$

in degree of a vertex:

The *inDegree* of a vertex is the number of arcs entering that vertex.

$$\mathrm{inDegree}[X] == \lambda\, r : X \leftrightarrow X \bullet \mathrm{outDegree}(r^\sim)$$

degree of a vertex:

The *degree* of a vertex is the number of arcs leaving and entering that vertex.

$$\mathrm{degree}[X] == \lambda\, r : X \leftrightarrow X \bullet \mathrm{outDegree}(\mathrm{symmetricClosure}\ r)$$

Laws

Law 31.9 *outDegree* is a total function (similarly for *inDegree* and *degree*).

$$[X] \vdash \text{outDegree}[X] \in (X \leftrightarrow X) \to X \nrightarrow \mathbb{N}$$

Law 31.10 A homogeneous function is a graph where the out degree is at most 1

$$[X] \vdash X \nrightarrow X = \{\ r : X \leftrightarrow X \mid \text{ran}(\text{outDegree}\, r) \subseteq \{0, 1\}\ \}$$

•

Chapter 32

Extending to infinite sets

We can use continuity and completeness arguments to extend the definition of a function from a finite to an infinite domain. We use this technique here to complete the definition of distributed arithmetic operators over infinite sets, from their definitions over finite sets.

- make a function complete over an infinite set: makeComplete
- complete distributed sum: Σ
- complete distributed product: Π

Making a function complete on a set

Intent

makeComplete is designed to be used on a function that is defined on some chain of subsets of real numbers, to turn it into a function on the union of the members of the chain.

Definition

$$
\begin{array}{|l}
[X] \\
\hline
\mathrm{makeComplete} : (\mathbb{P}\, X \pfun \mathbb{A}) \pfun (\mathbb{P}\, X \pfun \mathbb{A}) \\
\hline
\forall f : \mathbb{P}\, X \pfun \mathbb{R} \bullet \\
\quad \{\ a : \mathbb{P}\, X;\ y : \mathbb{R} \mid \\
\qquad \forall \delta : \mathbb{R}_+ \bullet \\
\qquad\quad \exists b : \mathrm{dom}\, f \mid b \subseteq a \bullet \\
\qquad\qquad \forall b_0 : \mathrm{dom}\, f \mid b \subseteq b_0 \subseteq a \bullet \\
\qquad\qquad\quad \mathrm{abs}(y - f\ b_0) \leq \delta\ \} \\
\subseteq \mathrm{makeComplete}\, f \\
\hline
\end{array}
$$

Typically *makeComplete* is applied to to a function whose domain is finite sets, and the result is a similar function with its domain extended to cover infinite sets. The technique used is the familiar one used to give meaning to the sums of infinite series in the absolutely convergent case. To see how it works, consider two cases

1. $a \in \operatorname{dom} f$
   We can then choose $b = a$, which forces $b_0 = a$, so $\operatorname{abs}(y - f\ b_0) < \delta$ implies $\operatorname{abs}(y - f\ a) < \delta$ for all δ, so $y = f\ a$. This shows us that $f \subseteq \operatorname{makeComplete} f$ and it is also clear by considering any other choice of b that no other value of y can fit.
2. $a \notin \operatorname{dom} f$
   We choose b to be a large subset of a within $\operatorname{dom} f$. b_0 then ranges over even larger sets. If the application of f to these sets converges, the result is that convergent value. Otherwise a is not in the domain of the result.

Examples

We use *makeComplete* in the definition of the distributed arithmetic operators Σ and Π below, to define their value for the infinite case.

Laws

Law 32.1 Making a function complete does not change it on its original domain.

$$[X]\ f : \mathbb{P}\, X \nrightarrow \mathbb{R} \vdash f \subseteq \operatorname{makeComplete} f$$

---•

Complete distributed sum

Intent

The operation of finite distributed sum is made complete.

Definition

$$[L]$$
$$\Sigma \ : (L \pfun \mathbb{A}) \pfun \mathbb{A}$$
$$\text{makeComplete}\, +\!/ \subseteq \Sigma$$

The use of the function $+/$ in the definition covers all finite labelled sets, then the use of *makeComplete* extends to cover infinite sets where the sum is 'absolutely convergent'. The application of this function may be pronounced *add up*.

Several Z authors have used a function with approximately this meaning, typically calling it $+/$ or Σ, but without formally defining it.

Examples

1. The sum of all the powers, for $abs\ x < 1$, both conventionally, and in Z, is:

$$\sum_{m=0}^{\infty} x^m = \frac{1}{1-x}$$

$$x : \mathbb{R} \mid \text{-1} < x < 1 \vdash \Sigma\ (\ \lambda\, m : \mathbb{N} \bullet x \mathbin{**} m\ ) = 1 \div (1 - x)$$

Laws

Law 32.2 Any sum formed with $+/$ can be equally well formed with Σ

$$[L]\ f : L \pfun \mathbb{A} \vdash +\!/\, f = \Sigma\, f$$

•

Complete distributed product

Intent

The operation of finite distributed product is made complete.

Definition

$$[L]$$
$$\Pi \ : (L \pfun \mathbb{A}) \pfun \mathbb{A}$$
$$\text{makeComplete} \, */ \subseteq \Pi$$

The use of the function $*/$ in the definition covers all finite labelled sets, then the use of *makeComplete* extends to cover infinite sets where the product is 'absolutely convergent'. The application of this function may be pronounced *multiply up*.

A few Z authors have used a function with approximately this meaning, typically calling it $*/$ or Π, but without formally defining it.

Examples

1. The product of odd squares, both conventionally, and in Z, is:

$$\prod_{m=1}^{\infty} \left(1 - \frac{1}{(2m+1)^2}\right) = \pi/4$$

$$\vdash \Pi \ (\ \lambda \, m : \mathbb{N}_1 \bullet 1 - 1 \div (2 * m + 1) ** 2 \ ) = \pi \div 4$$

2. The product of powers of powers, for $abs\ x < 1$, both conventionally, and in Z, is:

$$\prod_{m=0}^{\infty} \left(1 + x^{2^m}\right) = \frac{1}{1-x}$$

$$x : \mathbb{R} \mid \text{-1} < x < 1 \vdash \Pi \ (\ \lambda \, m : \mathbb{N} \bullet 1 + x ** 2 ** m \ ) = 1 \div (1 - x)$$

Laws

Law 32.3 Any sum formed with $*/$ can be equally well formed with Π

$$[L] \ f : L \pfin \mathbb{A} \vdash */ f = \Pi \ f$$

•

Chapter 33

Powers and Trigonometry

In this chapter we show how to use some of our earlier definitions to build up the definitions of the trigonometric functions.

- factorial
- power function: $(\_ ** \_)$
- exponential function: exp
- natural logarithm: ln
- common logarithm: log
- sine, cosine
- pi: π

Factorial

Intent

factorial n is the product of all the natural numbers from one to n.

Definition

function $(\text{factorial}\_)$

$$\text{factorial}\_ : \mathbb{A} \pfun \mathbb{A}$$

$$(\ \lambda\, n : \mathbb{N} \bullet \prod (\text{id}(1 \upto n)) \ ) \subseteq (\text{factorial}\_)$$

factorial n is conventionally written $n!$, but the ! symbol is reserved for other uses in Z.

Examples

1. *factorial* $0 = 1$
2. *factorial* $1 = 1$
3. *factorial* $2 = 2$
4. *factorial* $3 = 6$
5. *factorial* $4 = 24$

Laws

Law 33.1 The factorial function is a total function on the natural numbers; it is a total injection on the positive naturals.

$$\vdash \mathbb{N} \lhd (\text{factorial}\_) \in \mathbb{N} \to \mathbb{N}$$
$$\vdash \mathbb{N}_1 \lhd (\text{factorial}\_) \in \mathbb{N}_1 \rightarrowtail \mathbb{N}$$

Power function, integer exponent

Intent

We define the power function to cover all cases for real operands where a real result is defined and where that result is the principal value in the sense of complex variable theory. Thus where there is a choice of sign in the result, as when calculating $4{*}{*}(1 \div 2)$ for example, the function defines only the positive choice, yielding 2 and not -2.

We define the power function in two parts. First we define its value for an integer exponent (here), which we use in defining the exponential function, which in turn allows us to define the power function for a real exponent (later).

Definition

$$\text{function } 45 \text{ rightassoc } (\_ {*}{*} \_)$$

$$
\begin{array}{|l}
\_ ** \_ : \mathbb{A}^{2\times} \pfun \mathbb{A} \\
\hline
(\{0\} \times \mathbb{R}_+) \cup (\mathbb{R}_\pm \times \mathbb{Z}) \subseteq \mathrm{dom}(\_ ** \_) \\
\forall\, x : \mathbb{R} \bullet x ** 1 = x \\
\forall\, y : \mathbb{R}_+ \bullet 0 ** y = 0 \\
\forall\, x : \mathbb{R}_\pm;\ i : \mathbb{Z} \bullet x **(i+1) = x * x ** i
\end{array}
$$

The power function associates to the right and binds more strongly than multiplication. This ensures the expected behaviour, for example, that $x ** n ** 2 = x^{n^2}$ and $x * y ** i = x * y^i$.

———————————————•

Exponential function and natural logarithm

Intent

The exponential function is defined using a power series. In conventional notation it is:

$$e^x = 1 + \sum_{n=1}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

The natural logarithm is the inverse of the exponential.

Definition

exponential function:

$$\mathrm{function}\ (\exp \_)$$

$$
\begin{array}{|l}
\exp \_ : \mathbb{A} \pfun \mathbb{A} \\
\hline
(\ \lambda\, x : \mathbb{R} \bullet 1 + \Sigma\ (\ \lambda\, n : \mathbb{N}_+ \bullet x ** n \div \mathrm{factorial}\, n\ )\ ) \subseteq (\exp \_)
\end{array}
$$

The mathematical constant e is deliberately not defined under that name for fear of confusion with other uses of that letter in a Z specification. It can be immediately defined if needed as being equal to *exp* 1.

natural logarithm:

$$\mathrm{function}\ (\ln \_)$$

$$\ln\_ : \mathbb{A} \nrightarrow \mathbb{A}$$
$$\{\ x : \mathbb{R}_+;\ y : \mathbb{R} \mid x = \exp y\ \} \subseteq (\ln\_)$$

Laws

Law 33.2 The exponential function is a bijection from the reals to the positive reals.

$$\vdash \mathbb{R} \lhd (\exp\_) \in \mathbb{R} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{R}_+$$
$$\vdash \mathbb{R}_+ \lhd (\ln\_) \in \mathbb{R}_+ \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{R}$$

Law 33.3 The positive reals under multiplication are isomorphic to the reals under addition, where the isomorphism is the natural logarithm.

$$x, y : \mathbb{R} \vdash \exp(x + y) = \exp x * \exp y$$
$$x, y : \mathbb{R}_+ \vdash \ln x + \ln y = \ln(x * y)$$

---•

Power function, completed

Intent

The two sets of axioms given here and earlier for the power function together define its value for all cases of real operands where a real result is defined and where that result is the principal value in the sense of complex variable theory. So we have a negative result only where we have a negative first argument and an odd integer exponent.

Definition

$$\mathbb{R}_+ \times \mathbb{R} \subseteq \mathrm{dom}(\_ ** \_)$$
$$\forall\, x : \mathbb{R}_+;\ y : \mathbb{R} \bullet x ** y = \exp(y * \ln x)$$

---•

Common logarithm

Intent

The common logarithm is the inverse of the power function.

Definition

function $(\log\_)$

$$\log\_ : \mathbb{A} \nrightarrow \mathbb{A} \nrightarrow \mathbb{A}$$

$$\mathbb{R}_+ \setminus \{1\} \subseteq \mathrm{dom}(\log\_)$$
$$\forall\, base : \mathbb{R}_+ \setminus \{1\} \bullet \{\ x : \mathbb{R}_{\pm};\ y : \mathbb{R} \mid x = base ** y\ \} \subseteq \log_{base}$$

Examples

1. $\log_2\ 32 = 5$
2. $\log_{10}\ 1000 = 3$
3. Common logarithm to base e is the natural logarithm.

$$x : \mathbb{R}_+ \vdash \log_{\exp 1}\ x = \ln x$$

•

Sine and cosine

Intent

The sine function can be defined using a power series. In conventional notation it is:

$$\sin x = \sum_{n=0}^{\infty} \frac{(\text{-}1)^n x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$$

Similarly, the power series for the cosine function is

$$\cos x = 1 + \sum_{n=1}^{\infty} \frac{(\text{-}1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots$$

Definition

sine:

function $(\sin \_)$

$$\begin{array}{|l} \sin \_ : \mathbb{A} \pfun \mathbb{A} \\ \hline (\ \lambda\, x : \mathbb{R} \bullet \\ \quad \Sigma\ (\ \lambda\, n : \mathbb{N} \bullet \\ \quad\quad ((\text{-}1) ** n) * (x **(2 * n + 1)) \div \text{factorial}(2 * n + 1)\ )\ ) \\ \subseteq (\sin \_) \end{array}$$

cosine:

function $(\cos \_)$

$$\begin{array}{|l} \cos \_ : \mathbb{A} \pfun \mathbb{A} \\ \hline (\ \lambda\, x : \mathbb{R} \bullet \\ \quad 1 + \Sigma\ (\ \lambda\, n : \mathbb{N}_+ \bullet \\ \quad\quad ((\text{-}1) ** n) * (x **(2 * n)) \div \text{factorial}(2 * n)\ )\ ) \\ \subseteq (\cos \_) \end{array}$$

pi:

$$\pi == \min\{\ x : \mathbb{R}_+ \mid \sin x = 0\ \}$$

Further mathematical functions can easily be defined in similar ways to the above.

Laws

Law 33.4 Sine and cosine are total functions on the reals.

$$\vdash \mathbb{R} \lhd (\sin \_) \in \mathbb{R} \tfun \{\ x : \mathbb{R} \mid \text{-}1 \leq x \leq 1\ \}$$
$$\vdash \mathbb{R} \lhd (\cos \_) \in \mathbb{R} \tfun \{\ x : \mathbb{R} \mid \text{-}1 \leq x \leq 1\ \}$$

Law 33.5 Some rational approximations to π are

$$\vdash 223 \div 71 < \pi < 355 \div 113 < 22 \div 7$$

---•

Chapter 34

Streams and sequences

One useful special case of a labelling function is where the labels are *totally ordered*. The most common totally ordered labelling chosen is some consecutive segment of the natural numbers under 'less than'. Functions with such an ordered domain are called *streams*: $i \mathinner{..} \infty \to X$ in the infinite case, or $i \mathinner{..} j \to X$ in the finite case. The special case where $i = 1$ are called *sequences*.

A relation or function designed mainly to take a sequence as an argument can often still be meaningful if that argument is instead a more general relation; it may not require all the properties of sequences. For example:

- it may not require the indices to start at 1, but could be a stream, allowing them start at any natural number: $\{0 \mapsto a, 1 \mapsto b, 2 \mapsto c\}$
- it may not require the indices to be contiguous, but could allow 'gaps' $\{3 \mapsto a, 5 \mapsto b, 9 \mapsto c\}$
- it may not require the indices to be natural numbers, but could allow any totally ordered labelling set $\{-5 \mapsto a, \pi \mapsto b, 9 \mapsto c\}$
- it may not require the indices to be ordered at all, but could be defined for any function
- it may not require the sequence to be a single-valued function, but could be defined for any relation

So, when we define relations designed mainly for work with sequences, if they are capable of being used in more general contexts, we give more general definitions, and do not restrict the arguments to be sequences.

In this chapter we define streams and sequences. In the following chapters we define how to construct them from functions with an ordered domain, orders on streams and sequences, and relations for manipulating streams and sequences.

- streams, non-empty and injective: $\mathrm{stream}\, X$, $\mathrm{stream}_1\, X$, $\mathrm{istream}\, X$, $\mathrm{istream}_1\, X$

- sequences, non-empty and injective: $\mathrm{sequence}\, X$, $\mathrm{sequence}_1\, X$, $\mathrm{isequence}\, X$, $\mathrm{isequence}_1\, X$
- finite streams, non-empty and injective: $\mathrm{str}\, X$,$\mathrm{str}_1\, X$, $\mathrm{istr}\, X$, $\mathrm{istr}_1\, X$
- finite sequences, non-empty and injective: $\mathrm{seq}\, X$, $\mathrm{seq}_1\, X$, $\mathrm{iseq}\, X$, $\mathrm{iseq}_1\, X$
- shifting base: $(\_ \,\mathrm{shift}\, \_)$
- stream and sequence displays

General Streams

Intent

Streams may be finite or infinite. Their domain consists of a contiguous segment of the integers with a particular starting point (they can run up to infinity, but cannot run up from minus infinity).

Definition

streams:

$$\mathrm{generic}\,(\mathrm{stream}\,\_)$$
$$\mathrm{stream}\, X == \{\, f : \mathbb{Z} \pfun X \mid \exists\, m : \mathbb{Z} \bullet \forall\, n : \mathrm{dom}\, f \bullet m \leq n \land m \mathinner{..} n \subseteq \mathrm{dom}\, f \,\}$$

non-empty streams:

$$\mathrm{stream}_1\, X == \mathrm{stream}\, X \setminus \{\varnothing\}$$

injective streams:

If a stream is also an injection, each element in the stream is distinct.

$$\mathrm{generic}\,(\mathrm{istream}\,\_)$$
$$\mathrm{istream}\, X == \mathrm{stream}\, X \cap (\mathbb{Z} \pinj X)$$

non-empty injective streams:

$$\mathrm{istream}_1\, X == \mathrm{istream}\, X \setminus \{\varnothing\}$$

Examples

1. $(\ \lambda\, i : \mathbb{Z} \mid \text{-}5 < i \bullet i * i\ ) = {}_{\text{-}4}\langle 16, 9, 4, 1, 0, 1, 4, \ldots \rangle \in \operatorname{stream} \mathbb{N}$
2. $(\ \lambda\, i : \mathbb{N} \bullet \text{-}i\ ) = {}_{0}\langle 0, \text{-}1, \text{-}2, \text{-}3, \ldots \rangle \in \operatorname{istream} \mathbb{Z}$

Laws

Law 34.1 The domain of an infinite stream:

$$[X]\ s : \operatorname{stream} X \mid \neg\ \text{finite}\, s \vdash \exists\, i : \mathbb{Z} \bullet \operatorname{dom} s = \{\ j : \mathbb{Z} \mid i \leq j\ \}$$

Law 34.2 Restricting a stream to an integer range yields a finite stream. Restricting a stream to an integer range starting at 1 yields a finite sequence.

$$[X]\ s : \operatorname{stream} X;\ i, j : \mathbb{Z} \vdash (i \mathinner{..} j) \lhd s \in \operatorname{str} X$$
$$[X]\ s : \operatorname{stream} X;\ i : \mathbb{Z} \mid s = \varnothing \lor 1 \in \operatorname{dom} s \vdash (1 \mathinner{..} i) \lhd s \in \operatorname{seq} X$$

Law 34.3 The domain of a non-empty stream has a minimum element:

$$[X]\ s : \operatorname{stream}_1 X \vdash \operatorname{dom} s \in \operatorname{dom} \min$$

•

General Sequences

Intent

A sequence is a stream that starts at 1. Again, sequences may be finite or infinite. Their domain consists of an initial segment of the non-zero natural numbers.

Definition

sequences:

$$\text{generic}\ (\text{sequence}\ \_)$$
$$\operatorname{sequence} X == \{\ f : \mathbb{N}_1 \pfun X \mid \forall\, n : \operatorname{dom} f \bullet 1 \mathinner{..} n \subseteq \operatorname{dom} f\ \}$$

non-empty sequences:

$$\operatorname{sequence}_1 X == \operatorname{sequence} X \setminus \{\langle\ \rangle\}$$

injective sequences:

If a sequence is also an injection, each element in the sequence is distinct.

$$\text{generic}\,(\text{isequence}\,\_)$$
$$\text{isequence}\,X == \text{sequence}\,X \cap (\mathbb{N}_1 \rightarrowtail X)$$

non-empty injective sequences:

$$\text{isequence}_1\,X == \text{isequence}\,X \setminus \{\langle\,\rangle\}$$

Examples

1. $(\ \lambda\, n : \mathbb{N}_1 \bullet n \operatorname{div} 2\ ) = \langle 0, 1, 1, 2, 2, 3, 3, \ldots\rangle \in \text{sequence}\,\mathbb{N}$
2. $(\ \lambda\, n : \mathbb{N}_1 \bullet 1 - n\ ) = \langle 0, \text{-}1, \text{-}2, \text{-}3, \ldots\rangle \in \text{isequence}\,\mathbb{Z}$
3. The identity function on natural numbers is a non-empty injective sequence: $\text{id}\,\mathbb{N}_1 \in \text{isequence}_1\,\mathbb{N}_1$

Laws

Law 34.4 The domain of an infinite sequence is all the non-zero naturals:

$$[X]\ s : \text{sequence}\,X \mid \neg\ \text{finite}\,s \vdash \text{dom}\,s = \mathbb{N}_1$$

Law 34.5 Sequences are streams

$$[X]\ \vdash \text{sequence}\,X \subseteq \text{stream}\,X$$

---•

Finite streams

Definition

finite streams:

$$\text{generic}\,(\text{str}\,\_)$$
$$\text{str}\,X == \text{stream}\,X \cap (\mathbb{Z} \pfun X)$$

non-empty finite streams:

$$\text{str}_1\,X == \text{str}\,X \setminus \{\varnothing\}$$

finite injective streams:

$$\text{generic}\,(\text{istr}\,\_)$$
$$\text{istr}\,X == \text{istream}\,X \cap (\mathbb{Z} \pfun X)$$

non-empty finite injective streams:

$$\text{istr}_1\,X == \text{istr}\,X \setminus \{\varnothing\}$$

Examples

1. $\{0 \mapsto a, 1 \mapsto b, 2 \mapsto a\} = {}_0\langle a, b, a\rangle \in \text{str}\,X$
2. $(\ \lambda\, i : \text{-}2 \mathinner{..} 4 \bullet i * i\ ) = {}_{\text{-}2}\langle 4, 1, 0, 1, 4, 9, 16\rangle \in \text{str}\,\mathbb{N}$
3. $(\ \lambda\, n : 3 \mathinner{..} 10 \bullet \sqrt{n}\ ) = {}_3\langle \sqrt{3}, 2, \sqrt{5}, \sqrt{6}, \sqrt{7}, \sqrt{8}, 3, \sqrt{10}\rangle \in \text{istr}\,\mathbb{R}$

Laws

Law 34.6 The domain of a finite stream is an integer range.

$$[X]\ s : \text{str}\,X \vdash \exists\, i : \mathbb{Z} \bullet \text{dom}\,s = i \mathinner{..} (i - 1 + \#s)$$

•

Finite sequences

Definition

finite sequences:

$$\text{generic}\,(\text{seq}\_)$$
$$\text{seq}\,X == \text{sequence}\,X \cap (\mathbb{N}_1 \pfun X)$$

non-empty finite sequences:

$$\text{seq}_1\,X == \text{seq}\,X \setminus \{\langle\,\rangle\}$$

finite injective sequences:

$$\text{generic}\,(\text{iseq}\_)$$
$$\text{iseq}\,X == \text{isequence}\,X \cap (\mathbb{N}_1 \pfun X)$$

non-empty finite injective sequences:

$$\text{iseq}_1\,X == \text{iseq}\,X \cap \text{seq}_1\,X$$

Examples

1. $\{1 \mapsto a, 2 \mapsto b, 3 \mapsto a\} = \langle a, b, a\rangle \in \text{seq}\,X$
2. $(\ \lambda\, n : 1 \mathinner{..} 4 \bullet n \div 2\ ) = \langle 1 \div 2, 1, 3 \div 2, 2\rangle \in \text{seq}\,\mathbb{Q}$
3. $(\ \lambda\, n : 1 \mathinner{..} 10 \bullet \sqrt{n}\ ) = \langle 1, \sqrt{2}, \ldots, 3, \sqrt{10}\rangle \in \text{iseq}\,\mathbb{R}$
4. The identity function on some initial segment of the natural numbers is a non-empty finite injective sequence: $\forall\, n : \mathbb{N}_1 \bullet \text{id}(1 \mathinner{..} n) \in \text{iseq}_1\,\mathbb{N}_1$

Laws

Law 34.7 The domain of a finite sequence is the integer range from 1 to the length of the sequence.

$$[X]\ s : \text{seq}\,X \vdash \text{dom}\,s = 1 \mathinner{..} \#s$$

•

Shifting the base of a stream

Definition

function 60 leftassoc (_ shift _)

$\_\,\mathrm{shift}\,\_\ [X] == \lambda\, s : \mathrm{stream}\, X;\ i : \mathbb{Z} \bullet \{\ j : \mathrm{dom}\, s \bullet j + i \mapsto s\ j\ \}$

Examples

1. ${}_j\langle a, b, a\rangle\,\mathrm{shift}\,0 = {}_j\langle a, b, a\rangle$
2. ${}_j\langle a, b, a\rangle\,\mathrm{shift}\,i = {}_{(j+i)}\langle a, b, a\rangle$
3. ${}_j\langle a, b, a\rangle\,\mathrm{shift}(\text{-}j) = {}_0\langle a, b, a\rangle$
4. ${}_j\langle a, b, a\rangle\,\mathrm{shift}(\text{-}j + 1) = \langle a, b, a\rangle$

•

Stream and sequence displays

Definition

Stream displays have the index of the first element subscripted, followed by the stream elements in angle brackets.

function $(\_\langle , , \rangle)$

$\_\langle , , \rangle\ [X] == \lambda\, i : \mathbb{Z};\ s : \mathrm{seq}\, X \bullet s\,\mathrm{shift}(i - 1)$

Sequence displays have the elements in angle brackets.

function $(\langle , , \rangle)$

$\langle , , \rangle\ [X] == \lambda\, s : \mathrm{seq}\, X \bullet s$

Examples

1. $\{1 \mapsto a, 2 \mapsto b, 3 \mapsto a\} = \langle a, b, a\rangle$
2. $\{0 \mapsto a, 1 \mapsto b, 2 \mapsto a\} = {}_0\langle a, b, a\rangle$
3. $\{\text{-}2 \mapsto a, \text{-}1 \mapsto b, 0 \mapsto a\} = {}_{\text{-}2}\langle a, b, a\rangle$

•

Chapter 35

Constructing streams and sequences

We can construct streams and sequences by concatenating other streams and sequences. Also, given some arbitrary labelled set, we can convert that set into a sequence, provided the labelling domain is *enumerable* (can be written down in order).

This chapter provide the machinery for such constructions.

- concatenation: $(\_ \frown \_)$
- enumerable orders
- enumerable chains
- enumerate
- form a sequence from a labelled set: formSequence
- squash

Concatenation

Intent

Concatenation allows us to combine two streams to form a third, in such a way that it preserves the order of elements within each stream, and results in all the elements in the second stream appearing after all those in the first.

Definition

function 30 leftassoc $(\_ \frown \_)$

$$
\begin{array}{|l}
\hline
[X] \\
\_ \frown \_ : (\mathbb{A} \pfun X)^{2\times} \pfun (\mathbb{A} \pfun X) \\
\hline
\{\ s, t : \mathrm{stream}\, X;\ n : \mathbb{N} \mid \varnothing \in \{s, t\} \\
\qquad \lor \mathrm{finite}\, s \land n = \#s \land \min(\mathrm{dom}\, s) = \min(\mathrm{dom}\, t) \bullet \\
\qquad (s, t) \mapsto s \cup t\, \mathrm{shift}\, n\ \} \\
\quad \subseteq (\_ \frown \_) \\
\hline
\end{array}
$$

We provide a *declaration* wide enough for all numbered sets.

Our *definition* covers only the case where the arguments are streams, and, if neither is the empty stream, then the first stream is finite and the two streams have the same starting point. Sequences concatenated onto the end of finite sequences satisfy this constraint, with the minimum element being 1.

We provide a definition that extends the domain of concatenation as widely as possible without compromising other potential definitions. For example, there are (at least) four possible extensions to more general numbered sets that may be appropriate in different circumstances: the first set maintains its indexing, or the first set's indices are squashed to a contiguous set of integers; the second set's indices are all incremented by the same amount (related somehow to the last index of the first set), or the second set's indices are squashed. These would all be compatible extensions of our definition.

So we choose to define concatenation only for the case of a common minimum domain element, because if we arbitrarily chose the minimum of one of the arguments, we would require an arbitrary special case for when that argument is empty. Similarly, we define the result only for streams (contiguous domain values) because, if we try to use the spacing of the domain elements in some way, we would require arbitrary special cases when one or both arguments are singletons.

Examples

1. $\langle a, c, e\rangle \frown \langle b, d, f\rangle = \langle a, c, e, b, d, f\rangle$
2. $\{5 \mapsto 1, 6 \mapsto 2\} \frown (\ \lambda\, n : 5 \mathinner{..} 7 \bullet n \mathbin{**} 2\ ) = {}_5\langle 1, 2, 25, 36, 49\rangle$
3. $\langle 0\rangle \frown \mathrm{id}\, \mathbb{N}_1 = (\ \lambda\, n : \mathbb{N}_1 \bullet n - 1\ )$
4. $\{0 \mapsto a, 1 \mapsto b\} \frown \{0 \mapsto c\} = \{0 \mapsto a, 1 \mapsto b, 2 \mapsto c\}$

Laws

Law 35.1 For sequences, concatenation reduces to the simpler definition:

$$[X]\ s : \operatorname{seq} X;\ t : \operatorname{sequence} X \vdash s \frown t = s \cup t \operatorname{shift} \#s$$
$$[X]\ s : \operatorname{sequence} X \vdash s \frown \langle\ \rangle = s$$

Law 35.2 Stream concatenation is associative, and has identity element $\varnothing$. Hence it forms a monoid (§23) over finite streams.

$$[X]\ i : \mathbb{Z} \vdash \exists \operatorname{Monoid}[\mathbb{A} \leftrightarrow X] \bullet g = \{\ s : \operatorname{str} X \mid s = \varnothing \vee \min(\operatorname{dom} s) = i\ \} \wedge (\_ \diamond \_) = (\_ \frown \_) \wedge e = \varnothing$$

Law 35.3 The length of concatenated finite streams is the sum of the components' lengths.

$$[X]\ s, t : \operatorname{str} X \vdash \#(s \frown t) = \#s + \#t$$

Law 35.4 Concatenation is suffix-order preserving (§26) on its first argument, and prefix-order preserving on its second argument.

$$[X]\ s, t, u : \operatorname{seq} X \mid s \operatorname{suffix} t \vdash s \frown u \operatorname{suffix} t \frown u$$
$$[X]\ s, t, u : \operatorname{seq} X \mid s \operatorname{prefix} t \vdash u \frown s \operatorname{prefix} u \frown t$$

The order-preserving law about bounds (law 26.36) can be specialised for $f = s \operatorname{prefix} \_$ and for $f = \_ \operatorname{suffix} s$.

•

Enumerable order

Intent

An *enumerable ordering* is a total order where every element of the order has only a finite number of other elements 'less than' it. Hence the elements can be 'written down' in order (and hence, made into a sequence).

Definition

enumerable orders:

$$\text{generic}\,(\text{enumerableOrder}\,\_)$$
$$\text{enumerableOrder}\,X == \text{totalOrder}\,X \cap \text{locallyFiniteIn}\,X$$

reflexive enumerable orders:

$$\text{generic}\,(\text{reflexiveEnumerableOrder}\,\_)$$
$$\text{reflexiveEnumerableOrder}\,X == \text{enumerableOrder}\,X \cap \text{reflexive}\,X$$

irreflexive enumerable orders:

$$\text{generic}\,(\text{irreflexiveEnumerableOrder}\,\_)$$
$$\text{irreflexiveEnumerableOrder}\,X == \text{enumerableOrder}\,X \cap \text{irreflexive}\,X$$

Examples

Law 35.5 The natural numbers under 'less than' form an enumerable order

$$\vdash (\_ \leq \_) \cap \mathbb{N}^{2\times} \in \text{reflexiveEnumerableOrder}\,\mathbb{N}$$
$$\vdash (\_ < \_) \cap \mathbb{N}^{2\times} \in \text{irreflexiveEnumerableOrder}\,\mathbb{N}$$

The well order $1 \prec 3 \prec 5 \prec \ldots \prec 2 \prec 4 \prec 6 \prec \ldots$ is not an enumerable order. There are infinitely many numbers before 2 in this order.

Laws

Law 35.6 Any reflexive enumerable order is a well order.

$$[X] \vdash \text{reflexiveEnumerableOrder}\,X \subseteq \text{wellOrder}\,X$$

Law 35.7 Any finite total order is enumerable.

$$[X]\ r : \text{totalOrder}\,X \mid \text{finite}\,r \vdash r \in \text{enumerableOrder}\,X$$

---•

Enumerable chains

Intent

An enumerable chain is an enumerably ordered set with respect to a relation.

Definition

$$\begin{aligned}&\text{enumerableChain}[X] == \\ &\qquad \lambda\, r : X \leftrightarrow X \bullet \{\ a : \mathbb{P}\, X \mid r \cap a^{2\times} \in \text{enumerableOrder}\, a\ \} \\ &\text{enumerableChain}_1[X] == \lambda\, r : X \leftrightarrow X \bullet \text{enumerableChain}\, r \setminus \{\varnothing\}\end{aligned}$$

If we want to refer to some set a that is enumerably ordered by some relation r, we declare it as $a : \text{enumerableChain}\, r$. The full relation r may or may not be an enumerable order itself.

Examples

1. $\{\{a, b\}, \{a\}, \{a, b, c, d\}, \varnothing\} \in \text{enumerableChain}(\_ \subseteq \_)$
2. The upper bounds of prefixes of non-empty sequences form an enumerable chain.
3. The lower bounds of prefixes of non-empty sequences form a non-empty enumerable chain.
4. A chain like $1, 3, 5, 7, \ldots, 2, 4, 6, \ldots$ is not enumerable

Laws

Law 35.8 *enumerableChain* is a total function on homogeneous relations.

$$\begin{aligned}&[X] \vdash \text{enumerableChain}[X] \in (X \leftrightarrow X) \to \mathbb{P}\, \mathbb{P}\, X \\ &[X] \vdash \text{enumerableChain}_1[X] \in (X \leftrightarrow X) \to \mathbb{P}\, \mathbb{P}_1\, X\end{aligned}$$

Law 35.9 If r is an enumerable order, any subset of X is an enumerable chain.

$$[X]\ r : \text{enumerableOrder}\, X \vdash \text{enumerableChain}\, r = \mathbb{P}\, X$$

•

Enumeration of an enumerable chain

Intent

Convert an enumerable chain into a sequence by using the enumeration order.

Definition

$$\text{enumerate}[X] == \lambda\, r : \text{reflexive}\, X \bullet \lambda\, a : \text{enumerableChain}\, r \bullet \\ \{\ x : a \bullet \#(a \lhd r \rhd \{x\}) \mapsto x\ \}$$

Examples

$$\#\{a, b, c, d\} = 4 \Rightarrow \\ \text{enumerate}(\_ \subseteq \_)\{\{a, b\}, \{a\}, \{a, b, c, d\}, \varnothing\} = \\ \langle \varnothing, \{a\}, \{a, b\}, \{a, b, c, d\} \rangle$$

Laws

Law 35.10 *enumerate* is a total function on reflexive relations.

$$[X] \vdash \text{enumerate}[X] \in \text{reflexive}\, X \to \mathbb{P}\, X \nrightarrow \text{sequence}\, X$$

•

Forming a sequence from a labelled set

Intent

Form a sequence from a labelled set, where the labels are enumerable, by enumerating them.

Definition

$$\text{formSequence}[L, X] == \\ \lambda\, r : \text{reflexive}\, L \bullet \\ \lambda\, f : L \nrightarrow X \mid \text{dom}\, f \in \text{enumerableChain}\, r \bullet \\ (\text{enumerate}\, r(\text{dom}\, f)) \fatsemi f$$

Examples

$$\#\{a, b, c, d\} = 4 \Rightarrow \\ \quad \text{formSequence}(\_ \subseteq \_)\{\{a, b\} \mapsto w, \{a\} \mapsto x, \{a, b, c, d\} \mapsto y, \varnothing \mapsto z\} \\ \quad\quad = \langle z, x, w, y \rangle$$

Laws

Law 35.11 *formSequence* is a total function on reflexive relations.

$$[L, X] \ \vdash \text{formSequence} \in \text{reflexive}\, L \rightarrow (L \nrightarrow X) \nrightarrow \text{sequence}\, X$$

•

Squashing a sequence from a numbered set

Intent

Form a sequence from a labelled set, where the labels are numeric and enumerable.

Definition

$$[X]$$

$$\text{squash} : (\mathbb{A} \nrightarrow X) \nrightarrow \text{sequence}\, X$$

$$\text{formSequence}(\ (\_ \leq \_) \cap \mathbb{R}^{2\times}\ ) \subseteq \text{squash}$$

squash is typically used in the deletion of elements from a sequence, to renumber the remaining elements.

Examples

1. $\text{squash}\{3 \mapsto a, \text{-}2 \mapsto b, 7 \mapsto c, \pi \mapsto d\} = \langle b, a, d, c \rangle$

Laws

Law 35.12 Any function whose domain is the union of a finite set of reals and a set of natural numbers can be squashed:

$$[X] \vdash \{\, f : \mathbb{R} \pfun X \mid (\ \exists\, a : \mathbb{F}\,\mathbb{R};\ b : \mathbb{P}\,\mathbb{N} \bullet \operatorname{dom} f = a \cup b\ )\,\} \subseteq \operatorname{dom} \mathrm{squash}$$

Law 35.13 Only a sequence is left unchanged by *squash*.

$$[X]\ s : \operatorname{dom} \mathrm{squash} \vdash s \in \mathrm{sequence}\, X \Leftrightarrow s = \mathrm{squash}\, s$$

•

Chapter 36

Prefix, suffix and infix orders

The relations *prefix*, *suffix* and *infix* formalise the property of one stream being a contiguous 'part of' another. A prefix is a contiguous 'front part' of a stream; a suffix is a contiguous 'back part' of a stream; an infix is a contiguous 'mid part' of a stream.

Since these three relations are (pre-)orders, we naturally think about greatest lower bounds and least upper bounds. If these bounds were to be used frequently, it would be worth defining some short name for them; if they are used just occasionally, they can be referred to as glb(_ prefix _), and so on, as here.

- prefix, suffix relations
- infix relations
- prefix lower and upper bounds
- suffix lower and upper bounds
- infix lower and upper bounds

Prefix and suffix relations

Intent

Formalise the property of one stream being a contiguous 'part of' another, at the beggining or at the end.

Definition

A stream s is a prefix of a stream t precisely when there is a third stream u that may be concatenated onto s to give t. u may be empty, in which case s and t are

equal. (An infinite stream is a prefix of an infinite stream if they are equal.)

$$\text{relation } (\_ \text{ prefix } \_)$$
$$\_ \text{prefix} \_\ [X] == \{ s, t : \text{stream } X \mid \exists\, u : \text{stream } X \mid (s, u) \mapsto t \in (\_ \frown \_) \}$$

A stream s is a suffix of a stream t precisely when there is a third stream u onto which s may be concatenated to give t. u may be empty, in which case s and t are equal. (An infinite stream is a suffix of another infinite stream if they are equal, or if a finite stream may be added to the front of one to yield the other.)

$$\text{relation } (\_ \text{ suffix } \_)$$
$$\_ \text{suffix} \_\ [X] == \{ s, t : \text{stream } X \mid \exists\, u : \text{stream } X \mid (u, s) \mapsto t \in (\_ \frown \_) \}$$

Examples

1. $\langle a, c, e \rangle \text{ prefix} \langle a, c, e, b, d, f \rangle$
2. ${}_n\langle a, c, e \rangle \text{ prefix } {}_n\langle a, c, e, b, d, f \rangle$
3. $\langle b, d, f \rangle \text{ suffix} \langle a, c, e, b, d, f \rangle$
4. ${}_n\langle b, d, f \rangle \text{ suffix } {}_n\langle a, c, e, b, d, f \rangle$
5. $\langle 1, 2, 1, 2, 1, 2, \ldots \rangle \text{ suffix} \langle 1, 1, 1, 2, 1, 2, 1, 2, \ldots \rangle$
6. let $s == \langle 1, 2, 1, 2, 1, 2, \ldots \rangle;\ t == \langle 2, 1, 2, 1, 2, 1, \ldots \rangle$ •
   $s \text{ suffix } t \land t \text{ suffix } s \land s \neq t$

Laws

Law 36.1 The prefix relation provides an order on streams. The suffix relation provides preorders on streams, and orders on finite streams.

$$[X] \ \vdash (\_ \text{ prefix } \_) \cap (\text{stream } X)^{2\times} \in \text{reflexiveOrder}(\text{stream } X)$$
$$[X] \ \vdash (\_ \text{ suffix } \_) \cap (\text{stream } X)^{2\times} \in \text{preorder}(\text{stream } X)$$
$$[X] \ \vdash (\_ \text{ suffix } \_) \cap (\text{str } X)^{2\times} \in \text{reflexiveOrder}(\text{str } X)$$

Law 36.2 If a stream s is a prefix of a stream t, then it is a subset of t. A sequence s is a prefix of a sequence t precisely when it is a subset of t.

$$[X] \ s, t : \text{stream } X \mid s \text{ prefix } t \vdash s \subseteq t$$
$$[X] \ s, t : \text{sequence } X \vdash s \text{ prefix } t \Leftrightarrow s \subseteq t$$

Law 36.3 The finite sequence s is a prefix of the finite sequence t if it consists of the first n items of t; s is a suffix of t if it consists of t with the first n items removed.

$$[X]\ s, t : \operatorname{seq} X \vdash s \operatorname{prefix} t \Leftrightarrow s = (1 \mathinner{..} \# s) \upharpoonright t$$
$$[X]\ s, t : \operatorname{seq} X \vdash s \operatorname{suffix} t \Leftrightarrow s = (1 \mathinner{..} \# t - \# s) \downharpoonright t$$

Law 36.4 If s is a prefix of t, then s is compatible with t.

$$[X]\ s, t : \operatorname{stream} X \mid s \operatorname{prefix} t \vdash s \approx t$$

Law 36.5 The set of prefixes of suffixes equals the set of suffixes of prefixes.

$$[X]\ \vdash (\_ \operatorname{prefix} \_) \fatsemi (\_ \operatorname{suffix} \_) = (\_ \operatorname{suffix} \_) \fatsemi (\_ \operatorname{prefix} \_)$$

Infix relations

Intent

Formalise the property of one stream being a contiguous internal 'part of' another, that is, being a prefix of a suffix of another.

Definition

$$\operatorname{relation} (\_ \operatorname{infix} \_)$$
$$\_ \operatorname{infix} \_\ [X] == (\_ \operatorname{prefix} \_) \fatsemi (\_ \operatorname{suffix} \_)$$

An infinite stream is an infix of another infinite stream if they are equal, or if a finite stream may be added to the front of one to yield the other.

Examples

1. $\langle e, b \rangle \operatorname{infix} \langle a, c, e, b, d, f \rangle$
2. ${}_n\langle e, b \rangle \operatorname{infix} {}_n\langle a, c, e, b, d, f \rangle$

Laws

Law 36.6 The infix relation provides preorders on streams, and orders on finite streams.

$$[X] \vdash (\_ \operatorname{infix} \_) \cap (\operatorname{stream} X)^{2\times} \in \operatorname{preorder}(\operatorname{stream} X)$$
$$[X] \vdash (\_ \operatorname{infix} \_) \cap (\operatorname{str} X)^{2\times} \in \operatorname{reflexiveOrder}(\operatorname{str} X)$$

Law 36.7 A stream s is an infix of a stream t precisely when there are streams u and v that may be concatenated before and after s to give t. Hence $u \frown s$ is a prefix of t, and $s \frown v$ is a suffix of t.

$$[X] \; s, t : \operatorname{stream} X \vdash$$
$$s \operatorname{infix} t \Leftrightarrow$$
$$(\; \exists u, v : \operatorname{stream} X \mid \{(u, s), (s, v)\} \subseteq \operatorname{dom}(\_ \frown \_) \bullet u \frown s \frown v = t \;)$$

Law 36.8 A sequence s is an infix of a sequence t precisely when there is another sequence u that may be concatenated with s to give a subset of t.

$$[X] \; s, t : \operatorname{sequence} X \vdash$$
$$s \operatorname{infix} t \Leftrightarrow (\; \exists u : \operatorname{sequence} X \mid (u, s) \in \operatorname{dom}(\_ \frown \_) \bullet u \frown s \subseteq t \;)$$

Law 36.9 In the definition of *infix*, if u is empty, s is also a prefix of t; so any prefix is an infix. Also, if $u \frown s = t$, s is also a suffix of t; so any suffix is also an infix.

$$[X] \vdash (\_ \operatorname{prefix} \_) \subseteq (\_ \operatorname{infix} \_)$$
$$[X] \vdash (\_ \operatorname{suffix} \_) \subseteq (\_ \operatorname{infix} \_)$$

Law 36.10 s is an infix of t if it consists of some n consecutive items of t.

$$[X] \; s, t : \operatorname{seq} X \vdash s \operatorname{infix} t \Leftrightarrow (\; \exists n : \mathbb{N} \bullet s = (n \mathinner{..} n + \# s) \rhd t \;)$$

Law 36.11 s is an infix of t if it is the suffix of some prefix of t; s is an infix of t if it is the prefix of some suffix of t.

$$[X] \; s, t : \operatorname{stream} X \vdash s \operatorname{infix} t \Leftrightarrow (\; \exists u : \operatorname{stream} X \bullet s \operatorname{suffix} u \land u \operatorname{prefix} t \;)$$
$$[X] \; s, t : \operatorname{stream} X \vdash s \operatorname{infix} t \Leftrightarrow (\; \exists v : \operatorname{stream} X \bullet s \operatorname{prefix} v \land v \operatorname{suffix} t \;)$$

•

Prefix lower bounds

Intent

The prefix lower bounds of a set of streams are all those streams that are prefixes of every stream in the set.

Laws

Law 36.12 The set of prefix lower bounds is a non-empty enumerable chain (§35) that always contains at least the empty stream.

$$[X]\ a : \mathbb{P}_1\,\mathrm{stream}\,X \vdash \\ \quad \langle\,\rangle \in \mathrm{lowerBound}(\_\,\mathrm{prefix}\,\_)\,a \in \mathrm{enumerableChain}_1(\_\,\mathrm{prefix}\,\_)$$

For example

$$\mathrm{lowerBound}(\_\,\mathrm{prefix}\,\_)\{\langle w,x,y\rangle,\langle w,x,z\rangle\} = \{\langle\,\rangle,\langle w\rangle,\langle w,x\rangle\}$$
$$\mathrm{lowerBound}(\_\,\mathrm{prefix}\,\_)\{\langle x\rangle,\langle y\rangle\} = \{\langle\,\rangle\}$$

■ **Law 36.13** The chain of prefix lower bounds has a greatest lower bound, the 'longest common prefix'.

$$[X]\ \vdash \mathbb{P}_1\,\mathrm{stream}\,X \subseteq \mathrm{dom}(\mathrm{glb}(\_\,\mathrm{prefix}\,\_))$$

For example

$$\mathrm{glb}(\_\,\mathrm{prefix}\,\_)\{\langle w,x,y\rangle,\langle w,x,z\rangle\} = \langle w,x\rangle$$
$$\mathrm{glb}(\_\,\mathrm{prefix}\,\_)\{\langle x\rangle,\langle y\rangle\} = \langle\,\rangle$$

●

Prefix upper bounds

Intent

The prefix upper bounds of a set of streams are all those streams that have as prefixes every stream in the set.

Laws

Law 36.14 The set of prefix upper bounds is an enumerable chain; this chain may be empty.

$$[X]\ a : \mathbb{P}_1\,\mathrm{stream}\,X \vdash \\ \quad \mathrm{upperBound}(\_\,\mathrm{prefix}\,\_)\,a \in \mathrm{enumerableChain}(\_\,\mathrm{prefix}\,\_)$$

For example

$$\mathrm{upperBound}(\_\,\mathrm{prefix}\,\_)\{\langle x,y\rangle,\langle x,y,z\rangle\} = \\ \{\langle x,y,z\rangle,\langle x,y,z,x\rangle,\langle x,y,z,y\rangle,\langle x,y,z,z\rangle,\ldots\} \\ \mathrm{upperBound}(\_\,\mathrm{prefix}\,\_)\{\langle x\rangle,\langle y\rangle\} = \varnothing$$

Law 36.15 The chain of prefix upper bounds is non-empty precisely when the argument is a non-empty prefix-chain, in which case it has a least upper bound.

$$[X]\ \vdash \mathrm{chain}_1(\_\,\mathrm{prefix}\,\_)[X] = \mathrm{dom}(\mathrm{lub}(\_\,\mathrm{prefix}\,\_)[X])$$

For example

$$\mathrm{lub}(\_\,\mathrm{prefix}\,\_)\{\langle x,y\rangle,\langle x,y,z\rangle\} = \langle x,y,z\rangle$$

Law 36.16 The prefix least upper bound is the longest stream.

$$[X]\ a : \mathrm{chain}_1(\_\,\mathrm{prefix}\,\_)[X] \vdash \mathrm{lub}(\_\,\mathrm{prefix}\,\_)\,a = \bigcup a$$

Suffix lower bounds

Intent

The suffix lower bounds of a set of streams are all those streams that are suffixes of every stream in the set.

Laws

Law 36.17 The set of suffix lower bounds is a non-empty enumerable chain that always contains at least the empty stream.

$$[X]\ a : \mathbb{P}_1\,\mathrm{stream}\,X \vdash \\ \quad \langle\,\rangle \in \mathrm{lowerBound}(\_\,\mathrm{suffix}\,\_)\,a \in \mathrm{enumerableChain}_1(\_\,\mathrm{suffix}\,\_)$$

For example

$$\mathrm{lowerBound}(\_\,\mathrm{suffix}\,\_)\{\langle y, w, x\rangle, \langle z, w, x\rangle\} = \{\langle\,\rangle, \langle x\rangle, \langle w, x\rangle\}$$
$$\mathrm{lowerBound}(\_\,\mathrm{suffix}\,\_)\{\langle x\rangle, \langle y\rangle\} = \{\langle\,\rangle\}$$

Law 36.18 The chain of suffix lower bounds has a greatest lower bound, the 'longest common suffix'.

$$[X]\ \vdash \mathbb{P}_1\,\mathrm{stream}\,X \subseteq \mathrm{dom}(\mathrm{glb}(\_\,\mathrm{suffix}\,\_))$$

For example

$$\mathrm{glb}(\_\,\mathrm{suffix}\,\_)\{\langle y, w, x\rangle, \langle z, w, x\rangle\} = \langle w, x\rangle$$
$$\mathrm{glb}(\_\,\mathrm{suffix}\,\_)\{\langle x\rangle, \langle y\rangle\} = \langle\,\rangle$$

Suffix upper bounds

Intent

The suffix upper bounds of a set of streams are all those streams that have as suffixes every stream in the set.

Laws

Law 36.19 The set of suffix upper bounds is an enumerable chain; this chain may be empty.

$$[X]\ a : \mathbb{P}_1\,\mathrm{stream}\,X \vdash \\ \quad \mathrm{upperBound}(\_\,\mathrm{suffix}\,\_)\,a \in \mathrm{enumerableChain}(\_\,\mathrm{suffix}\,\_)$$

For example

$$\mathrm{upperBound}(\_\,\mathrm{suffix}\,\_)\{\langle x, y\rangle, \langle z, x, y\rangle\} = \\ \{\langle z, x, y\rangle, \langle x, z, x, y\rangle, \langle y, z, x, y\rangle, \langle z, z, x, y\rangle, \ldots\}$$
$$\mathrm{upperBound}(\_\,\mathrm{suffix}\,\_)\{\langle x\rangle, \langle y\rangle\} = \varnothing$$

Law 36.20 The chain of suffix upper bounds is non-empty precisely when the argument is a non-empty suffix-chain, in which case it has a least upper bound.

$$[X]\ \vdash \mathrm{chain}_1(\_\,\mathrm{suffix}\,\_)[X] = \mathrm{dom}(\mathrm{lub}(\_\,\mathrm{suffix}\,\_)[X])$$

For example

$$\mathrm{lub}(\_\,\mathrm{suffix}\,\_)\{\langle x, y\rangle, \langle z, x, y\rangle\} = \langle z, x, y\rangle$$

Law 36.21 The suffix least upper bound is the longest stream.

$$[X]\ a : \mathrm{chain}_1(\_\,\mathrm{suffix}\,\_)[X] \vdash \mathrm{lub}(\_\,\mathrm{suffix}\,\_)\,a = \mathrm{rev}(\bigcup(\mathrm{rev}(\!|\ a\ |\!)))$$

•

Infix lower bounds

Intent

The infix lower bounds of a set of streams are all those streams that are infixes of every stream in the set.

Laws

Law 36.22 The set of infix lower bounds always contains at least the empty stream.

$$[X]\ a : \mathbb{P}_1\,\mathrm{stream}\,X \vdash \langle\,\rangle \in \mathrm{lowerBound}(\_\,\mathrm{infix}\,\_)\,a$$

For example

$$\mathrm{lowerBound}(\_\,\mathrm{infix}\,\_)\{\langle x, y, z\rangle, \langle x, y, y, z\rangle\} = \{\langle\,\rangle, \langle x\rangle, \langle y\rangle, \langle z\rangle, \langle x, y\rangle, \langle y, z\rangle\}$$

Law 36.23 The chain of infix lower bounds has a greatest lower bound, the 'longest common infix', for non-empty infix-chains.

$$[X]\ \vdash \mathrm{chain}_1(\_\,\mathrm{infix}\,\_)[X] \subseteq \mathrm{dom}(\mathrm{lub}(\_\,\mathrm{infix}\,\_)[X])$$

For example

$$\mathrm{glb}(\_\,\mathrm{infix}\,\_)\{\langle x\rangle, \langle z, x\rangle, \langle z, x, y\rangle\} = \langle x\rangle$$

Greatest lower bounds may also exist for non infix-chains.

For example

$$\mathrm{glb}(\_\,\mathrm{infix}\,\_)\{\langle x, y\rangle, \langle y, z\rangle\} = \langle y\rangle$$

•

Infix upper bounds

Intent

The infix upper bounds of a set of streams are all those streams that have as infixes every stream in the set.

Laws

Law 36.24 The set of infix upper bounds of a set of finite streams is a non-empty set. (The set of upper bounds contains at least every concatenation of all elements of a.)

$$[X]\ a : \mathbb{P}_1 \operatorname{str} X \vdash \operatorname{upperBound}(\_ \operatorname{infix} \_)\, a \neq \varnothing$$

For example

$$\begin{aligned}&\operatorname{upperBound}(\_ \operatorname{infix} \_)\{\langle x\rangle, \langle y, z\rangle\} = \\ &\qquad \{\langle x, y, z\rangle, \langle y, z, x\rangle, \langle z, x, y, z\rangle, \langle z, y, z, x\rangle, \langle x, z, y, z\rangle, \ldots\} \\ &\operatorname{upperBound}(\_ \operatorname{infix} \_)\{\langle x, y\rangle, \langle z, x, y\rangle\} = \\ &\qquad \{\langle z, x, y\rangle, \langle z, x, y, x\rangle, \langle x, z, x, y\rangle, \langle z, x, y, y\rangle, \ldots\} \\ &\operatorname{upperBound}(\_ \operatorname{infix} \_)\{\langle x, x, x, \ldots\rangle, \langle y, y, y, \ldots\rangle\} = \varnothing\end{aligned}$$

Law 36.25 The set of infix upper bounds has a least upper bound if the argument is a non-empty infix-chain.

$$[X]\ \vdash \operatorname{chain}_1(\_ \operatorname{infix} \_)[X] = \operatorname{dom}(\operatorname{lub}(\_ \operatorname{infix} \_)[X])$$

For example

$$\operatorname{lub}(\_ \operatorname{infix} \_)\{\langle x\rangle, \langle z, x, y\rangle\} = \langle z, x, y\rangle$$

•

Chapter 37

Manipulating streams and sequences

In this chapter we define relations designed mainly for work with streams or sequences. However, if they are capable of being used in more general contexts, we give more general definitions, and do not restrict the arguments to be streams or sequences.

In other cases the given definition applies only to streams or sequences, but we frame it to allow extension to more general cases by providing a weaker declaration and choosing as far as possible forms of definition which generalise easily. Sensible generalisations of the definitions should let the various laws remain true on the extended domains (for example, a sensible generalisation of the definition of *rev* should preserve the law that it is self-inverse).

- reverse a finite numbered set: *rev*
- head, tail, last, front
- extraction, coextraction: $(\_ \restriction \_)$, $(\_ \downarrow \_)$
- filtering, cofiltering: $(\_ \upharpoonright \_)$, $(\_ \downharpoonright \_)$
- paths

Reversal

Intent

Reverse the order that the range elements appear in a finite numbered set, preserving the domain.

Definition

$$
\begin{array}{|l}
\hline
[X] \\
\hline
\mathrm{rev} : (\mathbb{A} \nrightarrow X) \nrightarrow (\mathbb{A} \nrightarrow X) \\
\hline
(\ \lambda f : \mathbb{R} \twoheadrightarrow X \bullet \\
\quad \{\ x, y : \mathrm{dom}\, f \mid \#\{\ z : \mathrm{dom}\, f \mid z < x\ \} = \#\{\ z : \mathrm{dom}\, f \mid y < z\ \} \bullet \\
\qquad x \mapsto f\ y\ \}\ ) \\
\subseteq \mathrm{rev} \\
\hline
\end{array}
$$

Examples

1. $\mathrm{rev}\{\text{-}2 \mapsto a, 3 \mapsto b, \pi \mapsto c, 42 \mapsto d\} = \{\text{-}2 \mapsto d, 3 \mapsto c, \pi \mapsto b, 42 \mapsto a\}$
2. $\mathrm{rev}\ {}_n\langle a, b, c, d\rangle = {}_n\langle d, c, b, a\rangle$
3. $\mathrm{rev}\langle a, b, c, d\rangle = \langle d, c, b, a\rangle$

Laws

Law 37.1 Reversal is a bijection.

$$[X]\ \vdash (\mathbb{R} \twoheadrightarrow X) \lhd \mathrm{rev} \in (\mathbb{R} \twoheadrightarrow X) \rightarrowtail\!\!\!\!\!\twoheadrightarrow (\mathbb{R} \twoheadrightarrow X)$$

Law 37.2 Reversal is self-inverse.

$$[X]\ s : \mathbb{R} \twoheadrightarrow X \vdash \mathrm{rev}(\mathrm{rev}\, s) = s$$

Law 37.3 Reversal preserves the domain of the function.

$$[X]\ s : \mathbb{R} \twoheadrightarrow X \vdash \mathrm{dom}\, s = \mathrm{dom}(\mathrm{rev}\, s)$$

Law 37.4 Reversing the empty and singleton functions has no effect; reversing a concatenation is the same as concatenating the reversals in the other order.

$$[X]\ \vdash \mathrm{rev}\, \varnothing[\mathbb{R} \times X] = \varnothing[\mathbb{R} \times X]$$
$$[X]\ i : \mathbb{R};\ x : X \vdash \mathrm{rev}\{i \mapsto x\} = \{i \mapsto x\}$$
$$[X]\ s, t : \mathrm{str}\, X \mid (s, t) \in \mathrm{dom}(\_ \frown \_) \vdash \mathrm{rev}(s \frown t) = \mathrm{rev}\, t \frown \mathrm{rev}\, s$$

Law 37.5 Performing an operation pointwise on the members of a stream, then reversing it, yields the same result as reversing it first, then performing the operation.

$$[X, Y]\ s : \mathbb{R} \twoheadrightarrow X;\ f : X \to Y \vdash \mathrm{rev}(f \circ s) = f \circ (\mathrm{rev}\ s)$$

Law 37.6 For finite streams and sequences, reversal reduces to the simpler definition:

$$[X]\ s : \mathrm{str}\ X \vdash \mathrm{rev}\ s = \lambda\, n : \mathrm{dom}\ s \bullet s(\max(\mathrm{dom}\ s) - n + \min(\mathrm{dom}\ s))$$
$$[X]\ s : \mathrm{seq}\ X \vdash \mathrm{rev}\ s = \lambda\, n : \mathrm{dom}\ s \bullet s(\#s - n + 1)$$

———————————•

head, last, tail, front

Intent

We define some operations for taking apart streams and sequences.

Definition

head can be applied to any numbered set whose domain has a minimum element: it gives the value indexed by that minimum element.

$$\mathrm{head}[X] == \lambda\, s : \mathbb{A} \nrightarrow X \mid \mathrm{dom}\ s \in \mathrm{dom}\ \min \bullet s(\min(\mathrm{dom}\ s))$$

tail can be applied to any numbered set with an enumerable domain. It removes the head element and moves all the indices down one place.

$$\mathrm{tail}[X] == \lambda\, s : \mathbb{A} \nrightarrow X \mid \mathrm{dom}\ s \in \mathrm{enumerableChain}(\_ < \_) \bullet$$
$$\{\ x, y : \mathrm{dom}\ s \mid x < y \land \neg\ (\ \exists\, z : \mathrm{dom}\ s \bullet x < z < y\ )\ \bullet\ x \mapsto s\ y\ \}$$

last can be applied to any numbered set whose domain has a maximum element: it gives the value indexed by that maximum element.

$$\mathrm{last}[X] == \lambda\, s : \mathbb{A} \nrightarrow X \mid \mathrm{dom}\ s \in \mathrm{dom}\ \max \bullet s(\max(\mathrm{dom}\ s))$$

front can be applied to any numbered set with a domain totally ordered by 'less than'. It includes all elements except the last one, if any (that is, the one where there is no element with a greater index).

$$\mathrm{front}[X] == \lambda\, s : \mathbb{A} \pfun X \mid (\mathrm{dom}\, s)^{2\times} \cap (\_ < \_) \in \mathrm{totalOrder}(\mathrm{dom}\, s) \bullet \{\ p, q : s \mid p.1 < q.1 \bullet p\ \}$$

Laws

Law 37.7 For streams, *tail* and *front* reduce to the simpler definitions:

$$[X]\ s : \mathrm{stream}\, X \vdash \mathrm{tail}\, s = \{\ m, n : \mathrm{dom}\, s \mid n = m + 1 \bullet m \mapsto s\ n\ \}$$
$$[X]\ s : \mathrm{stream}\, X \vdash \mathrm{front}\, s = \{\ m, n : \mathrm{dom}\, s \mid n = m + 1 \bullet m \mapsto s\ m\ \}$$

Law 37.8 *head* is a partial surjection. Its domain is any numbered set whose domain has a minimum element; this includes all non-empty streams and sequences.

$$[X]\ \vdash \mathrm{head}[X] \in (\mathbb{A} \pfun X) \twoheadrightarrow X$$
$$[X]\ \vdash \mathrm{dom}\, \mathrm{head} = \{\ s : \mathbb{A} \pfun X \mid \mathrm{dom}\, s \in \mathrm{dom}\, \mathrm{min}\ \}$$
$$[X]\ \vdash \mathrm{stream}_1\, X \rhd \mathrm{head} \in \mathrm{stream}_1\, X \twoheadrightarrow X$$

Law 37.9 *tail* is a partial function. Its domain is any numbered set with an enumerable domain; this includes all streams and sequences.

$$[X]\ \vdash \mathrm{tail}[X] \in (\mathbb{A} \pfun X) \pfun (\mathbb{A} \pfun X)$$
$$[X]\ \vdash \mathrm{dom}\, \mathrm{tail} = \{\ s : \mathbb{A} \pfun X \mid \mathrm{dom}\, s \in \mathrm{enumerableChain}(\_ < \_)\ \}$$
$$[X]\ \vdash \mathrm{stream}\, X \rhd \mathrm{tail} \in \mathrm{stream}\, X \twoheadrightarrow \mathrm{stream}\, X$$

Law 37.10 *last* is a partial surjection. Its domain is any numbered set whose domain has a maximum element; this includes all non-empty finite streams and sequences.

$$[X]\ \vdash \mathrm{last}[X] \in (\mathbb{A} \pfun X) \twoheadrightarrow X$$
$$[X]\ \vdash \mathrm{dom}\, \mathrm{last} = \{\ s : \mathbb{A} \pfun X \mid \mathrm{dom}\, s \in \mathrm{dom}\, \mathrm{max}\ \}$$
$$[X]\ \vdash \mathrm{str}_1\, X \rhd \mathrm{head} \in \mathrm{str}_1\, X \twoheadrightarrow X$$

Law 37.11 *front* is a partial function. Its domain is any numbered set with a domain totally ordered by 'less than'; this includes all streams and sequences.

$$[X]\ \vdash \mathrm{front}[X] \in (\mathbb{A} \pfun X) \pfun (\mathbb{A} \pfun X)$$
$$[X]\ \vdash \mathrm{dom}\, \mathrm{front} = \{\ s : \mathbb{A} \pfun X \mid (\mathrm{dom}\, s)^{2\times} \cap (\_ < \_) \in \mathrm{totalOrder}(\mathrm{dom}\, s)\ \}$$
$$[X]\ \vdash \mathrm{stream}\, X \rhd \mathrm{front} \in \mathrm{stream}\, X \twoheadrightarrow \mathrm{stream}\, X$$

Law 37.12 For a non-empty sequence, the *head* is the element indexed by 1. For a non-empty finite sequence, the *last* is the element indexed by the largest index, which is the length of the sequence.

$$[X]\ s : \mathrm{sequence}_1\ X \vdash \mathrm{head}\ s = s\ 1$$
$$[X]\ s : \mathrm{seq}_1\ X \vdash \mathrm{last}\ s = s(\#s)$$

Law 37.13 The head element of a reversed stream is the last element of the original stream; the tail of a reversed stream is the reverse of the front of the original stream.

$$[X]\ s : \mathrm{str}_1\ X \vdash \mathrm{head}(\mathrm{rev}\ s) = \mathrm{last}\ s$$
$$[X]\ s : \mathrm{str}_1\ X \vdash \mathrm{last}(\mathrm{rev}\ s) = \mathrm{head}\ s$$
$$[X]\ s : \mathrm{str}_1\ X \vdash \mathrm{tail}(\mathrm{rev}\ s) = \mathrm{rev}(\mathrm{front}\ s)$$
$$[X]\ s : \mathrm{str}_1\ X \vdash \mathrm{front}(\mathrm{rev}\ s) = \mathrm{rev}(\mathrm{tail}\ s)$$

Law 37.14 For numbered sets with a maximum domain element, *front* is simply the same set with that maximum element removed:

$$[X]\ s : \mathbb{A} \pfun X \mid \mathrm{dom}\ s \in \mathrm{dom}\ \mathrm{max} \vdash \mathrm{front}\ s = \{\mathrm{max}(\mathrm{dom}\ s)\} \rhd s$$

Law 37.15 The *head* and *last* of a singleton sequence are simply that element. The *tail* and *front* of a singleton sequence are the empty sequence.

$$[X]\ x : X \vdash \mathrm{head}\langle x\rangle = \mathrm{last}\langle x\rangle = x$$
$$[X]\ x : X \vdash \mathrm{tail}\langle x\rangle = \mathrm{front}\langle x\rangle = \langle\ \rangle$$

Law 37.16 *tail* is prefix and suffix order-preserving (§26).

$$[X]\ s : \mathrm{seq}_1\ X;\ t : \mathrm{sequence}_1\ X \mid s\ \mathrm{prefix}\ t \vdash (\mathrm{tail}\ s)\ \mathrm{prefix}(\mathrm{tail}\ t)$$
$$[X]\ s, t : \mathrm{sequence}_1\ X \mid s\ \mathrm{suffix}\ t \vdash (\mathrm{tail}\ s)\ \mathrm{suffix}(\mathrm{tail}\ t)$$

Law 37.17 *front* is prefix and suffix order-preserving (§26).

$$[X]\ s, t : \mathrm{seq}_1\ X \mid s\ \mathrm{suffix}\ t \vdash (\mathrm{front}\ s)\ \mathrm{suffix}(\mathrm{front}\ t)$$
$$[X]\ s, t : \mathrm{seq}_1\ X \mid s\ \mathrm{suffix}\ t \vdash (\mathrm{front}\ s)\ \mathrm{suffix}(\mathrm{front}\ t)$$

Law 37.18 The *head* of a non-empty concatenation is the head of the first sequence; the *tail* of a non-empty concatenation is the tail of the first sequence concatenated with the second sequence. The *last* of a non-empty concatenation is the last of the second sequence; the *front* of a non-empty concatenation is the first sequence concatenated with the front of the second sequence.

$$[X]\ s : \mathrm{seq}_1\, X;\ t : \mathrm{sequence}\, X \vdash \mathrm{head}(s \frown t) = \mathrm{head}\, s$$

$$[X]\ s : \mathrm{seq}_1\, X;\ t : \mathrm{sequence}\, X \vdash \mathrm{tail}(s \frown t) = \mathrm{tail}\, s \frown t$$

$$[X]\ s : \mathrm{seq}\, X;\ t : \mathrm{seq}_1\, X \vdash \mathrm{last}(s \frown t) = \mathrm{last}\, t$$

$$[X]\ s : \mathrm{seq}\, X;\ t : \mathrm{sequence}_1\, X \vdash \mathrm{front}(s \frown t) = s \frown \mathrm{front}\, t$$

Law 37.19 Concatenating the head and tail of a non-empty stream yields the original stream. Concatenating the front and last of a non-empty finite stream yields the original stream.

$$[X]\ s : \mathrm{stream}_1\, X;\ n : \mathbb{Z} \mid n = \min(\mathrm{dom}\, s) \vdash s = {}_n\langle \mathrm{head}\, s \rangle \frown \mathrm{tail}\, s$$

$$[X]\ s : \mathrm{str}_1\, X;\ n : \mathbb{Z} \mid n = \min(\mathrm{dom}\, s) \vdash s = \mathrm{front}\, s \frown {}_n\langle \mathrm{last}\, s \rangle$$

•

Extraction and filtering

Intent

Form a new sequence by removing selected items from a sequence.

Definition

extraction, coextraction:

$$\mathrm{function}\ 40\ \mathrm{leftassoc}\ (\_ \restriction \_)$$

$$\mathrm{function}\ 40\ \mathrm{leftassoc}\ (\_ \lfloor \_)$$

$$[X]$$

$$\_ \restriction \_, \_ \lfloor \_ : \mathbb{P}\, \mathbb{A} \times (\mathbb{A} \pfun X) \pfun (\mathbb{A} \pfun X)$$

$$(\ \lambda\, a : \mathbb{P}\, \mathbb{A};\ f : \mathbb{A} \pfun X \mid a \lhd f \in \mathrm{dom}\, \mathrm{squash} \bullet \mathrm{squash}(a \lhd f)\ )$$
$$\quad \subseteq (\_ \restriction \_)$$

$$\forall\, a : \mathbb{P}\, \mathbb{A};\ f : \mathbb{A} \pfun X \mid (\mathbb{A} \setminus a, f) \in \mathrm{dom}(\_ \restriction \_) \bullet$$
$$\quad a \lfloor f = (\mathbb{A} \setminus a) \restriction f$$

Extraction takes a set of indices and a numbered set, retains only those tuples from the numbered set whose indices are in the index set, and squashes the result, to yield an sequence.

Coextraction is similar, except that it *removes* tuples whose indices are in the index set.

We define extraction only for squashable arguments. We define coextraction in terms of extraction, so any later extension of extraction automatically extends the definition of coextraction suitably.

filtering, cofiltering:

function 40 leftassoc $(\_ \upharpoonright \_)$
function 40 leftassoc $(\_ \downharpoonright \_)$

$$
\begin{array}{|l}
\hline
[X] \\
\_ \upharpoonright \_, \_ \downharpoonright \_ : (\mathbb{A} \pfun X) \times \mathbb{P}\, X \pfun (\mathbb{A} \pfun X) \\
\hline
(\ \lambda f : \mathbb{A} \pfun X;\ b : \mathbb{P}\, X \mid f \rhd b \in \operatorname{dom} \operatorname{squash} \bullet \operatorname{squash}(f \rhd b)\ ) \\
\quad \subseteq (\_ \upharpoonright \_) \\
\forall f : \mathbb{A} \pfun X;\ b : \mathbb{P}\, X \mid (f, X \setminus b) \in \operatorname{dom}(\_ \upharpoonright \_) \bullet \\
\quad f \downharpoonright b = f \upharpoonright (X \setminus b) \\
\hline
\end{array}
$$

Filtering is similar to extraction, except that the filter set is a set of sequence elements, rather than sequence indices.

Examples

1. $\{2, 3, 8\} \upharpoonright \langle a, b, c, d, e, f \rangle = \langle b, c \rangle$
2. $\{2, 4, 8\} \upharpoonright \{4 \mapsto a, 2 \mapsto b, 1 \mapsto c\} = \langle b, a \rangle$
3. $\{2, 3\} \downharpoonright \langle a, b, c, d, e, f \rangle = \langle a, d, e, f \rangle$
4. $\langle a, b, c, d, e, f \rangle \upharpoonright \{e, f, h\} = \langle e, f \rangle$
5. $\langle a, b, c, d, e, f \rangle \downharpoonright \{e, f, h\} = \langle a, b, c, d \rangle$

Laws

Law 37.20 Coextraction can be expressed in terms of *squash*. Cofiltering can be expressed in terms of *squash*.

$$[X]\ a : \mathbb{P}\,\mathbb{A};\ f : \mathbb{A} \nrightarrow X \mid a \lhd f \in \operatorname{dom} \operatorname{squash} \vdash a \lfloor f = \operatorname{squash}(a \lhd f)$$
$$[X]\ f : \mathbb{A} \nrightarrow X;\ b : \mathbb{P}\,X \mid f \rhd b \in \operatorname{dom} \operatorname{squash} \vdash f \rfloor b = \operatorname{squash}(f \rhd b)$$

Law 37.21 Extracting and filtering the empty sequence has no effect.

$$[X]\ a : \mathbb{P}\,\mathbb{A} \vdash a \upharpoonright \varnothing[\mathbb{A} \times X] = \varnothing[\mathbb{A} \times X] = a \downharpoonright \varnothing[\mathbb{A} \times X]$$
$$[X]\ b : \mathbb{P}\,X \vdash \langle\,\rangle \restriction b = \langle\,\rangle = \langle\,\rangle \lfloor b$$

Law 37.22 Extracting and filtering on the empty set retains nothing. Coextracting and cofiltering on the empty set removes nothing.

$$[X]\ f : \mathbb{A} \nrightarrow X \vdash \varnothing \upharpoonright f = \langle\,\rangle = f \restriction \varnothing$$
$$[X]\ s : \operatorname{sequence} X \vdash \varnothing \downharpoonright s = s = s \lfloor \varnothing$$

Law 37.23 Extraction retains everything if the extracting set contains all the sequence indices. Filtering retains everything if the filtering set contains all the sequence elements.

$$[X]\ a : \mathbb{P}\,\mathbb{A};\ f : \mathbb{A} \nrightarrow X \mid a \lhd f \in \operatorname{dom} \operatorname{squash} \vdash \operatorname{dom} f \subseteq a \Leftrightarrow a \upharpoonright f = f$$
$$[X]\ f : \mathbb{A} \nrightarrow X;\ b : \mathbb{P}\,X \mid f \rhd b \in \operatorname{dom} \operatorname{squash} \vdash \operatorname{ran} f \subseteq b \Leftrightarrow f \restriction b = f$$

Law 37.24 Filtering twice filters on the intersection.

$$[X]\ s : \operatorname{sequence} X;\ a, b : \mathbb{P}\,X \vdash (s \restriction a) \restriction b = s \restriction (a \cap b)$$

Law 37.25 Extracting and filtering preserve injectivity.

$$[X]\ a : \mathbb{P}\,\mathbb{A};\ f : \mathbb{A} \rightarrowtail X \mid a \lhd f \in \operatorname{dom} \operatorname{squash} \vdash a \upharpoonright f \in \mathbb{A} \rightarrowtail X$$
$$[X]\ a : \mathbb{P}\,\mathbb{A};\ f : \mathbb{A} \rightarrowtail X \mid a \lhd f \in \operatorname{dom} \operatorname{squash} \vdash a \downharpoonright f \in \mathbb{A} \rightarrowtail X$$
$$[X]\ f : \mathbb{A} \rightarrowtail X;\ b : \mathbb{P}\,X \mid f \rhd b \in \operatorname{dom} \operatorname{squash} \vdash f \restriction b \in \mathbb{A} \rightarrowtail X$$
$$[X]\ f : \mathbb{A} \rightarrowtail X;\ b : \mathbb{P}\,X \mid f \rhd b \in \operatorname{dom} \operatorname{squash} \vdash f \rfloor b \in \mathbb{A} \rightarrowtail X$$

Law 37.26 Extracting and filtering do not increase the length of the sequence.

$$[X]\ a : \mathbb{P}\,\mathbb{A};\ s : \operatorname{seq} X;\ b : \mathbb{P}\,X \vdash \max\{\#(a \upharpoonright s), \#(a \downharpoonright s), \#(s \upharpoonright b), \#(s \downharpoonright b)\} \leq \#s$$

Law 37.27 Filtering distributes through concatenation.

$$[X]\ s : \operatorname{seq} X;\ t : \operatorname{sequence} X;\ b : \mathbb{P}\,X \vdash (s \frown t) \upharpoonright b = (s \upharpoonright b) \frown (t \upharpoonright b)$$

$$[X]\ s : \operatorname{seq} X;\ t : \operatorname{sequence} X;\ b : \mathbb{P}\,X \vdash (s \frown t) \downharpoonright b = (s \downharpoonright b) \frown (t \downharpoonright b)$$

•

Paths and steps

Intent

A *path* in a homogeneous relation is a finite sequence of vertices, each adjacent pair in the sequence being connected by an arc in the relation.

The *steps* of a stream is the relation comprising the set of consecutive pairs of elements of the stream.

Definition

paths:

$$\operatorname{path}[X] == \lambda\, r : X \leftrightarrow X \bullet \{\ s : \operatorname{seq} X \mid \forall\, m, n : \operatorname{dom} s \mid m + 1 = n \bullet s\ m \mapsto s\ n \in r\ \}$$

non-empty paths:

$$\operatorname{path}_1[X] == \lambda\, r : X \leftrightarrow X \bullet \operatorname{path} r \setminus \{\varnothing\}$$

injective paths:

$$\operatorname{ipath}[X] == \lambda\, r : X \leftrightarrow X \bullet \operatorname{path} r \cap \operatorname{iseq} X$$

non-empty injective paths:

$$\operatorname{ipath}_1[X] == \lambda\, r : X \leftrightarrow X \bullet \operatorname{ipath} r \setminus \{\varnothing\}$$

non-trivial paths:

$$\mathrm{path}_2[X] == \lambda\, r : X \leftrightarrow X \bullet \{\ p : \mathrm{path}\, r \mid 2 \leq \#p\ \}$$

non-trivial injective paths:

$$\mathrm{ipath}_2[X] == \lambda\, r : X \leftrightarrow X \bullet \{\ p : \mathrm{ipath}\, r \mid 2 \leq \#p\ \}$$

steps:

$$\mathrm{steps}[X] == \lambda\, s : \mathrm{stream}\, X \bullet \{\ m, n : \mathrm{dom}\, s \mid m + 1 = n \bullet s\ m \mapsto s\ n\ \}$$

Examples

1. $\forall\, r : X \leftrightarrow X \bullet \langle\ \rangle \in \mathrm{path}\, r$
2. $\forall\, x : X;\ r : X \leftrightarrow X \bullet \langle x \rangle \in \mathrm{path}\, r$
3. $\{\langle 2, 4, 16, 256\rangle, \langle 3, 9, 81\rangle\} \subseteq \mathrm{path}\ \mathit{square}$
4. $\mathrm{steps}\langle\ \rangle = \varnothing$
5. $\mathrm{steps}\langle 2 \rangle = \varnothing$
6. $\mathrm{steps}\langle 2, 4, 16, 256\rangle = \{2 \mapsto 4, 4 \mapsto 16, 16 \mapsto 256\}$

Laws

Law 37.28 *path* is a total function.

$$[X] \vdash \mathrm{path} \in (X \leftrightarrow X) \to \mathbb{P}(\mathrm{seq}\, X)$$

Law 37.29 The relational iterations of a relation is the set of all (head, last) pairs of length $n + 1$.

$$[X]\ r : X \leftrightarrow X;\ n : \mathbb{N} \vdash r^n = \{\ s : \mathrm{path}\, r \mid \#s = n + 1 \bullet \mathrm{head}\, s \mapsto \mathrm{last}\, s\ \}$$

For example

$$\{3 \mapsto 9, 9 \mapsto 81, 81 \mapsto 6561\} \subseteq \mathit{square}$$
$$\langle 3, 9, 81, 6561\rangle \in \mathrm{path}\ \mathit{square}$$
$$3 \mapsto 6561 \in \mathit{square}^3$$

Law 37.30 The transitive closure of a relation is the set of all (head, last) pairs from all the non-trivial paths.

$$[X]\ r : X \leftrightarrow X \vdash r^+ = \{\ s : \mathrm{path}_2\, r \bullet \mathrm{head}\, s \mapsto \mathrm{last}\, s\ \}$$

Law 37.31 The reflexive transitive closure of a relation is the set of all (head, last) pairs from all the non-empty paths.

$$[X]\ r : X \leftrightarrow X \vdash r^* = \{\ s : \mathrm{path}_1\, r \bullet \mathrm{head}\, s \mapsto \mathrm{last}\, s\ \}$$

Law 37.32 The intransitive relations are those that have paths with (head, last) pairs also in the relation only if those paths are of length two.

$$[X]\ \vdash \mathrm{intransitive}\, X = \{\ r : X \leftrightarrow X \mid \forall\, s : \mathrm{path}_1\, r \mid \mathrm{head}\, s \mapsto \mathrm{last}\, s \in r \bullet \#s = 2\ \}$$

Law 37.33 The acyclic relations are those that have only injective paths.

$$[X]\ \vdash \mathrm{acyclic}\, X = \{\ r : X \leftrightarrow X \mid \mathrm{path}\, r = \mathrm{ipath}\, r\ \}$$

Law 37.34 The strongly connected relations are those that have a non-empty path from every domain vertex to every range vertex.

$$[X]\ \vdash \mathrm{stronglyConnected}\, X = \{\ r : X \leftrightarrow X \mid \forall\, x : \mathrm{dom}\, r;\ y : \mathrm{ran}\, r \bullet \exists\, p : \mathrm{path}_1\, r \bullet x = \mathrm{head}\, p \land y = \mathrm{last}\, p\ \}$$

Law 37.35 *steps* is the inverse of *path*

$$[X]\ s : \mathrm{seq}\, X \mid 2 \leq \#s \vdash s \in \mathrm{path}(\mathrm{steps}\, s)$$

Law 37.36 For a non-empty finite stream, the length of the stream is one more than the size of the induced relation. (So if p is a path, then $\#p$ is the number of vertices, which is one more than the number of links, or steps.)

$$[X]\ s : str_1 X \vdash \#s = 1 + \#(\mathrm{steps}\, s)$$

Law 37.37 The steps of a path comprise a subset of the original relation.

$$[X]\ r : X \leftrightarrow X \vdash \forall\, s : \mathrm{path}\, r \bullet \mathrm{steps}\, s \subseteq r$$

Law 37.38 For every non-empty path s in relation r, there is a non-empty injective path s' in r that beginnings and ends at the same place as s, but uses only some of the steps. (The injective path misses out any "loops" in the non-injective path.)

$$[X]\ r : X \leftrightarrow X \vdash \forall\, s : \mathrm{path}_1\, r \bullet \\ \exists\, s' : \mathrm{ipath}_1\, r \mid \mathrm{steps}\, s' \subseteq \mathrm{steps}\, s \bullet \mathrm{head}\, s' = \mathrm{head}\, s \land \mathrm{last}\, s' = \mathrm{last}\, s$$

———————————————•

Chapter 38

Sequenced families of sets

Earlier we introduced the functions $+/$ and $*/$, for distributed sum and product, and the function *distributeOverLabelledSet*, for general distribution of an abelian monoid. Now we relax the requirement that the monoid be abelian.

The minimum assumptions to allow meaningful distribution are that the function is a monoid, and that we are distributing over a finite sequence (since the sequence gives an order, the monoid does not have to be abelian). The more general function *distributeOverSeq* allows us to define distributed functions when the monoid is not abelian.

- distribute a monoid over a finite sequence: distributeOverSeq
- distributed overriding: $\oplus/$
- distributed composition: $\fatsemi/$, $\circ/$
- distributed concatenation: $\frown/$

APL uses the convention of modifying an infix function to show distribution across a finite sequence by appending the character '/' [Iverson 1962]. It is there referred to as 'reduction', and the name 'reduce' is also used in other languages such as Common Lisp [Steele, Jr. 1990]. In other literature on functional programming, the operation is sometimes referred to as 'folding' (see for example [Bird & Wadler 1988]).

Distributed override is also defined in [Hayes 1987, p60], and in [Sufrin 1986, 2.6.11] named with a 'fat' override symbol.

Distributed composition is also defined in [Hayes 1987, p15], and in [Sufrin 1986, 2.6.11] named with a 'fat' composition symbol.

Distributing a monoid over a sequence

Definition

$$
\begin{array}{l}
\text{distributeOverSeq}[X] == \\
\quad \{\ \text{Monoid}[X];\ \diamond/ : \text{seq}\, X \nrightarrow X \mid \\
\qquad\qquad \diamond/ \in \text{seq}\, g \rightarrow g \\
\qquad\qquad \land\ \diamond/\ \langle\ \rangle = e \\
\qquad\qquad \land\ (\ \forall x : g \bullet \diamond/\ \langle x \rangle = x\ ) \\
\qquad\qquad \land\ (\ \forall s, t : \text{seq}\, g \bullet \diamond/\ (s \frown t) = \diamond/\ s\ \diamond\ \diamond/\ t\ )\ \}
\end{array}
$$

Laws

Law 38.1 *distributeOverSeq* is a total function, that yields a total surjection on the semigroup set g.

$$
\begin{array}{l}
[X]\ \vdash \text{distributeOverSeq}[X] \in \text{Monoid}[X] \rightarrow \text{seq}\, X \nrightarrow X \\
[X]\ \text{Monoid}[X] \vdash \text{distributeOverSeq}\ \theta\ \text{Monoid}[X] \in \text{seq}\, g \twoheadrightarrow g
\end{array}
$$

Law 38.2 The relationship between *distributeOverSeq* and *distributeOverLabelledSet* is:

$$
\begin{array}{l}
[X]\ \text{AbelianMonoid}[X] \vdash \\
\quad \text{distributeOverSeq}\ \theta\ \text{AbelianMonoid}[X] \\
\qquad = \text{distributeOverLabelledSet}(\mathbb{N}_1, g, (\_ \diamond \_))
\end{array}
$$

Law 38.3 *distributeOverSeq* can be broken up as follows:

$$
\begin{array}{l}
[X]\ \vdash \\
\quad \text{distributeOverSeq}[X] = \\
\qquad \{\ \text{Monoid}[X];\ \diamond/\ : \text{seq}\, X \rightarrow X \mid \\
\qquad\quad \diamond/\ \langle\ \rangle = e \\
\qquad\quad \land\ (\ \forall s : \text{seq}_1\, X \bullet \\
\qquad\qquad \diamond/\ (\langle \text{head}\, s \rangle \frown \text{tail}\, s) = \text{head}\, s \diamond (\diamond/\ (\text{tail}\, s))\ )\ \}
\end{array}
$$

•

Distributed override

Intent

Distributed override acts on a sequence of relations, of which an important special case is when they are functions. The operation does an effective override on them, beginning to end.

Definition

$$\begin{aligned}\oplus/\ [X, Y] == \lambda\, s : \mathrm{sequence}(X \leftrightarrow Y) \bullet \\ \quad \{\ r : \bigcup(\mathrm{ran}\ s) \mid \exists\, n : \mathrm{dom}\ s \mid r \in s\ n \bullet \\ \quad\quad \forall\, m : \mathrm{dom}\ s \mid m > n \bullet r.1 \notin \mathrm{dom}(s\ m)\ \}\end{aligned}$$

The argument can be a possible infinite sequence of relations.

Laws

Law 38.4 Distributed override is a total surjection.

$$[X, Y]\ \vdash \oplus/ \in \mathrm{sequence}(X \leftrightarrow Y) \twoheadrightarrow (X \leftrightarrow Y)$$

Law 38.5 Overriding the empty sequence gives the empty relation; overriding the singleton sequence gives the element; overriding a length two sequence gives the two elements composed by the binary form of the operator.

$$\begin{aligned}&[X, Y]\ \vdash \oplus/\ \varnothing[\mathbb{A} \times (X \leftrightarrow Y)] = \varnothing[X \times Y] \\ &[X, Y]\ r : X \leftrightarrow Y \vdash \oplus/\ \langle r \rangle = r \\ &[X, Y]\ r, s : X \leftrightarrow Y \vdash \oplus/\ \langle r, s \rangle = r \oplus s\end{aligned}$$

Law 38.6 Many of the properties of binary overriding can be generalised to distributed overriding. Law 20.11, that the domain of an overriding is the union of the individual domains, generalises to:

$$[X, Y]\ s : \mathrm{seq}(X \leftrightarrow Y) \vdash \mathrm{dom}(\oplus/\ s) = \bigcup(\mathrm{ran}(\mathrm{dom} \circ s))$$

Law 38.7 Our definition of $\oplus/$ is bigger than that obtained by using *distributeOverSeq*:

$$[X, Y] \vdash$$
$$\quad \text{distributeOverSeq}(\!|\ g == X \leftrightarrow Y, \_ \diamond \_ == (\_ \oplus \_)[X, Y], e == \varnothing[X \times Y]\ |\!)$$
$$\qquad \subseteq \oplus/$$

Law 38.8 Complete distributed override is linked to maximal iteration, *do*, by relational iteration:

$$[X]\ r : X \leftrightarrow X \vdash \text{do}\, r = \oplus/(\ \lambda\, n : \mathbb{N} \bullet r^n\ )$$

Distributed composition

Intent

Distributed composition acts on a finite sequence of homogeneous relations, and forms the result of composing them all with each other, beginning to end. We can pronounce the application of this function as 'compose up'.

Definition

distributed relational composition:

$$\fatsemi/\ [X] ==$$
$$\quad \text{distributeOverSeq}(\!|\ g == X \leftrightarrow X, \_ \diamond \_ == (\_ \fatsemi \_)[X, X, X], e == \text{id}\, X\ |\!)$$

distributed functional composition:

$$\circ/\ [X] ==$$
$$\quad \text{distributeOverSeq}(\!|\ g == X \leftrightarrow X, \_ \diamond \_ == (\_ \circ \_)[X, X, X], e == \text{id}\, X\ |\!)$$

Laws

Law 38.9 Distributed relational composition is a total surjection.

$$[X]\ \vdash \fatsemi/\ [X] \in \text{seq}(X \leftrightarrow X) \twoheadrightarrow (X \leftrightarrow X)$$

Law 38.10 Composing the empty sequence gives the identity; composing the singleton sequence gives the element; composing a length two sequence gives the two elements composed by the binary form of the operator.

$$[X] \vdash \fatsemi/\ \langle\,\rangle = \mathrm{id}\, X$$
$$[X]\ r : X \leftrightarrow X \vdash \fatsemi/\ \langle r \rangle = r$$
$$[X]\ r, s : X \leftrightarrow X \vdash \fatsemi/\ \langle r, s \rangle = r \fatsemi s$$

Law 38.11 Some of the properties of binary composition can be generalised to distributed composition. Law 20.23, that inverting a composition is the same as composing the inversions in the other order, generalises to:

$$[X]\ s : \mathrm{seq}(X \leftrightarrow X) \vdash (\fatsemi/\ s)^{\sim} = \fatsemi/\ (\mathrm{rev}((\_^{\sim}) \circ s))$$

Law 38.12 Relational iteration can be expressed as a distributed relational composition.

$$[X]\ r : X \leftrightarrow X;\ n : \mathbb{N} \vdash r^n = \fatsemi/(\ \lambda\, i : 1 \mathinner{..} n \bullet r\ )$$

Similar laws to those given above for distributed relational composition apply to distributed functional composition.

———————————————•

Distributed concatenation

Definition

$$\begin{array}{l} \frown/[X] == \\ \quad \lambda\, s : \mathrm{sequence}(\mathrm{sequence}\, X) \mid \\ \qquad s \in \mathrm{seq}(\mathrm{sequence}\, X) \land \mathrm{ran}(\mathrm{front}\, s) \subseteq \mathrm{seq}\, X \\ \qquad \lor\ \mathrm{ran}\, s \subseteq \mathrm{seq}\, X \bullet \\ \quad \{\ m : \mathrm{dom}\, s;\ n : \mathbb{N}_+ \mid n \in \mathrm{dom}(s\ m) \bullet \\ \qquad \Sigma\ (\ \lambda\, i : 1 \mathinner{..} (m-1) \bullet \#(s\ i)\ ) + n \mapsto s\ m\ n\ \} \end{array}$$

The argument s can be a finite sequence of sequences where only the last of the constituent sequences is allowed to be infinite, or can be a possibly infinite sequence of finite sequences.

The application of this function may be pronounced ‘concatenate up’.

Laws

Law 38.13 Distributed concatenation is a partial surjection.

$$[X] \vdash \frown/[X] \in \operatorname{sequence}(\operatorname{sequence} X) \pfun \operatorname{sequence} X$$

Law 38.14 Many of the properties of binary concatenation can be generalised to distributed concatenation. Law 35.3, that the length of a concatenation is the sum of the individual lengths, generalises to:

$$[X] \; s : \operatorname{seq}\operatorname{seq} X \vdash \#(\frown/ \; s) = \Sigma \; (\# \circ s)$$

Law 38.15 Our definition of $\frown/$ is bigger than that obtained by using *distributeOverSeq*:

$$[X] \vdash \operatorname{distributeOverSeq}(\!| \; g == \operatorname{seq} X, \_ \diamond \_ == (\_ \frown \_)[X], e == \varnothing[\mathbb{A} \times X] \; |\!) \subseteq \frown/$$

Chapter 39

Bags

A set is an unordered collection of distinct elements. We model order using sequences. The usual way to model the number of occurrences of each element is to use a *bag*, or *multi-set*.

A natural way to model a bag in Z is to use a function with some numerical range, $X \pfun \mathbb{A}$. The domain is the bag elements, each mapped to the number of times it occurs in the bag.

[Hayes 1990] suggests that Z bags not be restricted to a natural number range (as they are in *ZRM*, [Spivey 1992]). Hayes widens the range to $\mathbb{Z}$, defines a rich set of bag operations, and gives a variety of examples of their use. We have started from this idea, and choose a wider bag domain still, in our case $\mathbb{R}$.

- bags, non-empty bags, finite bags: $\mathrm{bag}\, X$, $\mathrm{bag}_1\, X$, $\mathrm{fbag}\, X$
- number of elements in a bag: count
- bag sum: $(\_ \uplus \_)$
- building a bag: items
- bag display
- mean, median, mode
- prime factors

Bags

Intent

A bag is a function from items to non-zero real numbers. Each item is mapped to the number of times that item occurs in the bag (zero if it is not in the domain of the bag).

We choose to use a partial function, with zero not in the range, rather than a total function, to have the desirable property that $X \subseteq Y \Rightarrow \mathrm{bag}\, X \subseteq \mathrm{bag}\, Y$. This follows from the order-preserving property of $\nrightarrow$ (law 21.6), whereas $\rightarrow$ is not order preserving on its domain argument (law 21.14).

Hence a bag is not a suitable model in cases where we wish to distinguish an element being present zero times from an element being absent.

Definition

bags:

$$\mathrm{generic}\,(\mathrm{bag}\_)$$
$$\mathrm{bag}\ \ X == X \nrightarrow \mathbb{R}_{\pm}$$

non-empty bags:

$$\mathrm{bag}_1\, X == \mathrm{bag}\, X \setminus \varnothing$$

finite bags:

$$\mathrm{fbag}\, X == X \nrightarrow\!\!\!\!\!\!\rightarrow \mathbb{R}_{\pm}$$

number of elements in a bag:

We define a totalised version of the bag function, *count*, to give an always-applicable number of occurrences of an element in a bag (that is, including the case of zero).

$$\mathrm{count}[X] == \lambda\, b : \mathrm{bag}\, X \bullet (\ \lambda\, x : X \bullet 0\ ) \oplus b$$

If count is used on a bag on numbers, it may be advisable to instantiate its generic parameter explicitly, as for example count[$\mathbb{R}$], since otherwise the whole of $\mathbb{A}$ will be implied, which may not be what was wanted.

Examples

1. $\{\varnothing \mapsto 1, \{x, y, z\} \mapsto 2, \{x\} \mapsto 2, \{x, y\} \mapsto 3\} \in \mathrm{bag}\, \mathbb{P}\, X$
2. $\{\varnothing \mapsto \text{-}1, \{x, y, z\} \mapsto 2 \div 3, \{x\} \mapsto \pi, \{x, y\} \mapsto 3\} \in \mathrm{bag}\, \mathbb{P}\, X$
3. $\{\text{-}3 \mapsto 1, 3 \mapsto 2, \pi \mapsto 2, 2 \mapsto 3\} \in \mathrm{bag}\, \mathbb{R}$
4. $\mathrm{count}\{\varnothing \mapsto 1, \{x, y, z\} \mapsto 2, \{x\} \mapsto 2, \{x, y\} \mapsto 3\}\varnothing = 1$
5. $\mathrm{count}\{\varnothing \mapsto 1, \{x, y, z\} \mapsto 2, \{x\} \mapsto 2, \{x, y\} \mapsto 3\}\{y\} = 0$

Laws

Law 39.1 *count* is a total surjection.

$$[X] \vdash \text{count} \in \text{bag}\, X \twoheadrightarrow X \to \mathbb{R}$$

Law 39.2 If we apply *count* to a bag, then range subtract the zero elements, we restore the original bag.

$$[X]\ b : \text{bag}\, X \vdash (\text{count}\, b) \rhd \{0\} = b$$

•

Functions of a single bag

Intent

To apply any function $f : \mathbb{R} \to \mathbb{R}$ to bag frequencies we write $f \circ b$. So to multiply all bag frequencies by some constant $r : \mathbb{R}$ (known as *bag scaling*, $\otimes$, in *ZRM*) we write

$$(\ \lambda\, x : \mathbb{R} \bullet r * x\ ) \circ b$$

•

Functions of two bags

Intent

Apply a numeric operation to two bags.

Definition

bag compose:

To apply a numeric relation r to the frequencies of a pair of bags, b and c, we first convert the bags to total functions suitable for bicomposition, then bicompose, then convert the result back to a bag.

$$\begin{array}{l}\text{bagcompose}[X] == \\ \quad \lambda\, r : \mathbb{A}^{2\times} \leftrightarrow \mathbb{A} \bullet \lambda\, b, c : \text{bag}\, X \bullet \\ \qquad \text{bicompose}\, r(\text{count}\, b, \text{count}\, c) \rhd \{0\}\end{array}$$

sum of two bags:

$$\text{function } 30 \,\text{leftassoc}\,(\_ \uplus \_)$$
$$\_ \uplus \_\,[X] == \lambda\, b, c : \text{bag}\, X \bullet \text{bagcompose}(\_ + \_)(b, c)$$

Examples

1. $\{x \mapsto 4\} \uplus \{x \mapsto 5\} = \{x \mapsto 9\}$
2. $\{x \mapsto \text{-}4\} \uplus \{x \mapsto 5\} = \{x \mapsto 1\}$
3. $\{x \mapsto \text{-}4\} \uplus \{x \mapsto 4\} = \varnothing$
4. $\{x \mapsto \text{-}4\} \uplus \{y \mapsto 4\} = \{x \mapsto \text{-}4, y \mapsto 4\}$
5. To subtract bag frequencies of two bags, allowing negative results:

   $$\text{bagcompose}(\_ - \_)(b, c)$$

6. To subtract bag frequencies of two bags, rounding negative results up to zero (known as *bag subtraction*, $\uplus$, in *ZRM*):

   $$\text{bagcompose}(\ \lambda\, x, y : \mathbb{R} \bullet \max\{x - y, 0\}\ )(b, c)$$

7. To form a bag that, for each item, picks the larger frequency of its argument bags:

   $$\text{bagcompose}(\ \lambda\, x, y : \mathbb{R} \bullet \max\{x, y\}\ )(b, c)$$

8. To form a bag that, for each item, picks the smaller frequency of its argument bags:

   $$\text{bagcompose}(\ \lambda\, x, y : \mathbb{R} \bullet \min\{x, y\}\ )(b, c)$$

Laws

Law 39.3 $\uplus$ is a total surjection.

$$[X] \ \vdash (\_ \uplus \_)[X] \in (\text{bag}\, X)^{2\times} \twoheadrightarrow \text{bag}\, X$$

---•

Building a bag

Intent

Define *items*, to convert a labelled set of items (including a finite stream or sequence) into the corresponding bag (whose range is necessarily confined to natural numbers).

Definition

$$\text{items}[L, X] == \lambda f : L \pfun X \mid (\forall x : X \bullet \text{finite}(f \rhd \{x\})) \bullet (\lambda x : X \bullet \#(f \rhd \{x\}))$$

Compared with the version in *ZRM*, our definition has an extra generic parameter, since we allow any labelled family, not just a sequence.

Examples

1. $\text{items}\langle\rangle = \varnothing$
2. $\text{items}\langle a, b, a, b, c, a, b, c, a\rangle = \{a \mapsto 4, b \mapsto 3, c \mapsto 2\}$

Laws

Law 39.4 *items* is a function. Its domain is those labelled sets where each labelled element occurs only a finite number of times. Its range is bags whose ranges themselves are confined to natural numbers.

$$[L, X] \vdash \text{items}[L, X] \in (L \pfun X) \pfun \text{bag}\, X$$
$$[L, X] \vdash \text{dom}\,\text{items}[L, X] = \{ f : L \pfun X \mid \forall x : X \bullet f \rhd \{x\} \in \text{dom}\,\# \}$$
$$[L, X] \vdash \text{ran}\,\text{items}[L, X] = X \pfun \mathbb{N}_1$$

Law 39.5 The domain of the bag formed from a stream is the same as the range of the stream.

$$[X] \; s : \text{str}\, X \vdash \text{dom}(\text{items}\, s) = \text{ran}\, s$$

Law 39.6 Concatenating streams, then forming a bag, gives the same result as forming the individual bags, then adding them.

$$[X]\ s, t : \mathrm{str}\, X \vdash \mathrm{items}(s \frown t) = \mathrm{items}\, s \uplus \mathrm{items}\, t$$

Law 39.7 Two streams form the same bag precisely when they are permutations.

$$[X]\ s, t : \mathrm{str}\, X \vdash \mathrm{items}\, s = \mathrm{items}\, t \Leftrightarrow (\ \exists f : \mathrm{dom}\, s \rightarrowtail\!\!\!\!\twoheadrightarrow \mathrm{dom}\, t \bullet s = t \circ f\ )$$

———————————————•

Bag display

Intent

Write an explicit bag.

Definition

There are two simple ways to explicitly describe an arbitrary bag:

1. using a set display, for example: $\{x \mapsto 3, y \mapsto 2, z \mapsto 1\}$
2. for positive integer frequencies only, by applying *items* to a sequence display, for example: $\mathrm{items}\langle x, x, x, y, y, z\rangle$

The second of these is more economical when it applies at all and when the average bag frequency is less than 2, but for high bag frequencies only the first is viable. For example the bag $\{x \mapsto 3472, y \mapsto 2963, z \mapsto 1444\}$ is easily described as such, but not reasonably by the second method. The second method is not possible for negative and fractional bag frequencies.

In *ZRM* special bags brackets ⟦ and ⟧ are defined, equivalent in meaning to the second display method above, that is as *items* applied to a sequence display. *Standard* Z gives us the power to define such bag brackets by use of a template, if we so choose.

For example, as

$$\mathrm{function}\,(⟦, ,⟧)$$
$$⟦, ,⟧\ [X] == \lambda\, s : \mathrm{seq}\, X \bullet \mathrm{items}\, s$$

We chose not to make this definition here, because of its limited applicability and the existence of suitable alternatives. So the symbols ⟦ and ⟧ are still available for other uses.

Examples

1. $\text{items}\langle x, x, x, y, z, z\rangle = \{x \mapsto 3, y \mapsto 1, z \mapsto 2\}$
2. $\text{items}\langle x, x, z\rangle \uplus \text{items}\langle y, x\rangle = \text{items}\langle x, x, x, y, z\rangle$

•

Further bag operations

Intent

We provide examples of how further operations may be defined on bags.

Examples

1. To add up the total number of elements in a finite bag b, we write

$$sizebag[X] == \lambda\, b : \text{fbag}\, X \bullet +/\, b$$

2. To add up the total numeric value of a finite bag of numbers b:

$$sumbag == \lambda\, b : \text{fbag}\, \mathbb{R} \bullet +/(\ \lambda\, x : \text{dom}\, b \bullet x * (b\ x)\ )$$

3. To multiply up the total numeric value of a finite bag of numbers b:

$$productbag == \lambda\, b : \text{fbag}\, \mathbb{R} \bullet */(\ \lambda\, x : \text{dom}\, b \bullet x **(b\ x)\ )$$

Averages example

The *arithmetic mean* of a finite bag of real numbers can be defined as the total numeric value divided by the total number of elements:

$$mean == \lambda\, b : \text{fbag}\, \mathbb{R} \mid +/\, b \neq 0 \bullet +/(\ \lambda\, x : \text{dom}\, b \bullet x * (b\ x)\ ) \div +/\, b$$

This definition works for negative and fractional occurrences, too, to give a *weighted* mean.

1. $mean(\text{items}\langle 0, 3, 4, 1, 2, 2\rangle) = 2$
2. $x : \mathbb{R};\ y : \mathbb{R}_{\pm} \vdash mean\{x \mapsto y\} = x$

3. $x, x', y, y' : \mathbb{R} \mid y + y' \neq 0 \vdash \mathit{mean}\{x \mapsto y, x' \mapsto y'\} = (x * y + x' * y') \div (y + y')$
4. $\mathit{mean}\{2 \mapsto 1, 4 \mapsto 1\} = (2 * 1 + 4 * 1) \div (1 + 1) = 3$
5. $\mathit{mean}\{\text{-}2 \mapsto 1, \text{-}4 \mapsto 1\} = (\text{-}2 * 1 + \text{-}4 * 1) \div (1 + 1) = \text{-}3$
6. $\mathit{mean}\{2 \mapsto \text{-}1, 4 \mapsto \text{-}1\} = (2 * \text{-}1 + 4 * \text{-}1) \div (\text{-}1 + \text{-}1) = 3$
7. $\mathit{mean}\{2 \mapsto \text{-}2, 4 \mapsto 1\} = (2 * \text{-}2 + 4 * 1) \div (\text{-}2 + 1) = 0$

The *median* (middle) of a bag of totally ordered elements can be defined as:

$$\begin{aligned}\mathit{median}[X] == \lambda\, \mathit{to} : \mathrm{totalOrder}\, X \bullet \lambda\, b : X \nrightarrow \mathbb{R} \bullet \\ \{\ m : \mathrm{dom}\, b \mid \\ +/(\mathrm{successors}(\mathit{to} \setminus \mathrm{id}\, X) m \lhd b) \\ \leq +/(\mathrm{predecessors}(\mathit{to} \cup \mathrm{id}\, X) m \lhd b) \\ \land +/(\mathrm{predecessors}(\mathit{to} \setminus \mathrm{id}\, X) m \lhd b) \\ \leq +/(\mathrm{successors}(\mathit{to} \cup \mathrm{id}\, X) m \lhd b)\ \}\end{aligned}$$

This definition makes no restrictions on the frequencies (not even that they be non-zero). It says that for every median value m, the number of bag elements strictly greater than m (according to the order) do not exceed the number below and including m, and also the number of bag elements strictly less than m do not exceed the number above and including m.

When the generic parameter is (real) numbers, and total order is the usual order on those numbers, $\mathit{to} = (\_ < \_)[\mathbb{R}]$ or $\mathit{to} = (\_ \leq \_)[\mathbb{R}]$, this definition reduces to

$$\begin{aligned}\mathit{median}_{\mathbb{R}} == \lambda\, b : \mathbb{R} \nrightarrow \mathbb{R} \bullet \{\ m : \mathrm{dom}\, b \mid \\ +/(\{\ x : \mathbb{R} \mid m < x\ \} \lhd b) \leq +/(\{\ x : \mathbb{R} \mid x \leq m\ \} \lhd b) \\ \land +/(\{\ x : \mathbb{R} \mid x < m\ \} \lhd b) \leq +/(\{\ x : \mathbb{R} \mid m \leq x\ \} \lhd b)\ \}\end{aligned}$$

If the frequencies are all (and at least one) positive, the result is a set of one or two elements.

1. $\mathit{median}_{\mathbb{R}}\{x \mapsto 1\} = \{x\}$
2. $\mathit{median}_{\mathbb{R}}(\mathrm{items}\langle 0, 0, 2, 3, 7\rangle) = \{2\}$
3. $\mathit{median}_{\mathbb{R}}(\mathrm{items}\langle 0, 0, 2, 3, 7, 9\rangle) = \{2, 3\}$
4. $\mathit{median}_{\mathbb{R}}\{1 \mapsto 1 \div 4, 2 \mapsto 3 \div 4, \} = \{2\}$

If the frequencies are all non-negative, and at least one positive, as when one writes $\mathit{median}\ \mathit{to}(\mathrm{count}\, b)$, or $\mathit{median}(\_ < \_)(\mathrm{count}[\mathbb{R}] b)$ for a bag of numbers, the result may be a singleton set, or one containing two values corresponding to positive

frequencies together with all values of the type between those values corresponding to zero frequencies. This may allow appropriate interpolation.

If any frequencies are negative, the behaviour and its interpretation are left as an exercise for the reader.

The *mode* (most commonly occurring elements) of a bag of items can be defined as:

$$mode[X] == \lambda\, b : \operatorname{bag} X \bullet \{\ x : X \mid \forall\, y : X \bullet \operatorname{count} b\ y \leq \operatorname{count} b\ x\ \}$$

1. $mode\ \varnothing = \varnothing$
2. $mode(\operatorname{items}\langle 0, 3, 3, 1, 1, 2, 3\rangle) = \{3\}$
3. $mode(\operatorname{items}\langle \varnothing, \{x, y, z\}, \{x, y, z\}, \{x\}, \{x\}, \{x, y\}\rangle) = \{\{x\}, \{x, y, z\}\}$
4. $mode\{x \mapsto 1, y \mapsto 2\} = \{y\}$
5. $mode\{x \mapsto \text{-}1, y \mapsto \text{-}2\} = X \setminus \{x, y\}$
   since all the other elements are present zero times, more than either x or y

All of *sizebag*, *sumbag*, *productbag*, *mean* and *median to* are functions on finite sets which could be made arguments to *makeComplete*, thus extending them to apply to infinite sets where there is convergence.

Prime factors example

Any positive integer greater than one has a unique decomposition into prime factors. For example, $60 = 2*2*3*5$; $45 = 3*3*5$. In fact, unique factorisation into powers of primes also applies to all positive rationals, and to all rational powers of positive rationals. So we can represent such a number by the bag of its prime factors. Recall that *productbag* maps a bag of numbers to the product of those numbers; its inverse maps a number into the bag of its factors. We restrict it to those numbers that can be uniquely factorised, thuse ensuring that we get a function.

$$\text{factor} == productbag^{\sim} \rhd (\text{prime} \pfun \mathbb{Q})$$

$\text{factor}\, 60 = \{2 \mapsto 2, 3 \mapsto 1, 5 \mapsto 1\}$
$\text{factor}\, 45 = \{3 \mapsto 2, 5 \mapsto 1\}$
$\text{factor}\, 1 = \varnothing$
$\text{factor}(3 \div 4) = \{3 \mapsto 1, 2 \mapsto \text{-}2\}$
$\text{factor}(\sqrt{6}) = \{2 \mapsto 1 \div 2, 3 \mapsto 1 \div 2\}$

We can multiply two numbers by adding the relevant prime factors:

$$n, m : \mathbb{N} \vdash \text{factor}(n * m) = \text{factor}\, n \uplus \text{factor}\, m$$

$$\begin{aligned}
&60 * 45 \\
&\quad = \mathit{productbag}(\text{factor}\, 60 \uplus \text{factor}\, 45) \\
&\quad = \mathit{productbag}(\{2 \mapsto 2, 3 \mapsto 1, 5 \mapsto 1\} \uplus \{3 \mapsto 2, 5 \mapsto 1\}) \\
&\quad = \mathit{productbag}\{2 \mapsto 2, 3 \mapsto 3, 5 \mapsto 2\} \\
&\quad = 2 ** 2 * 3 ** 3 * 5 ** 2 \\
&\quad = 2700
\end{aligned}$$

We can raise a number to a power by using bag scaling, multiplying each occurrence in the bag of factors by that power:

$$n, m : \mathbb{N} \vdash \text{factor}(n ** m) = (\text{factor}\, n) \fatsemi (\ \lambda x : \mathbb{R} \bullet m * x\ )$$

$$\begin{aligned}
&\text{factor}(60 ** 3) = \{2 \mapsto 6, 3 \mapsto 3, 5 \mapsto 3\} \\
&\text{factor}(60 ** \text{-}1) = \{2 \mapsto \text{-}2, 3 \mapsto \text{-}1, 5 \mapsto \text{-}1\} \\
&\text{factor}(45 ** 2) = \{3 \mapsto 4, 5 \mapsto 2\} \\
&\text{factor}(45 ** 1 \div 2) = \{3 \mapsto 1, 5 \mapsto 1 \div 2\}
\end{aligned}$$

The greatest common divisor and least common multiple can be found from the factor representation:

$$\begin{aligned}
&\text{gcd} == \lambda\, n, m : \mathbb{N}_1 \bullet \\
&\qquad \mathit{productbag}(\text{bagcompose}(\ \lambda\, x, y : \mathbb{R} \bullet \min\{x, y\}\ )(\text{factor}\, n, \text{factor}\, m)) \\
&\text{lcm} == \lambda\, n, m : \mathbb{N}_1 \bullet \\
&\qquad \mathit{productbag}(\text{bagcompose}(\ \lambda\, x, y : \mathbb{R} \bullet \max\{x, y\}\ )(\text{factor}\, n, \text{factor}\, m))
\end{aligned}$$

$$\begin{aligned}
&\text{gcd}(60, 45) = \mathit{productbag}\{3 \mapsto 1, 5 \mapsto 1\} = 15 \\
&\text{lcm}(60, 45) = \mathit{productbag}\{2 \mapsto 2, 3 \mapsto 2, 5 \mapsto 1\} = 180
\end{aligned}$$

Two numbers are coprime if they have no factors in common:

$$\vdash (\text{coprime}\_) = \{\ n, m : \mathbb{N}_2 \mid \text{disjoint}\langle \text{dom}(\text{factor}\, n), \text{dom}(\text{factor}\, m)\rangle\ \}$$

———————————————————•

Part VI

Example Specifications

Chapter 40

Changing representations: a memory map

This chapter illustrates a use of streams and distributed concatenation.

There are many occasions where it is useful to swap between two representations of a data structure, one in terms of a stream of variable sized structures, one in terms of a stream of fixed sized primitive elements, often bytes or words. For example:

- Java byte code's Constant Pool contains a collection of structures, many of which contain indexes that point to other parts of the pool considered as an array of words, which indexes must point to the beginning of valid structures.
- Many low level assembly languages with variable length instructions have flow of control jumps to a memory location, not to an instruction.

Without assuming anything about the detailed internals of the structures and primitives, we can define a general way of mapping between them.

$$[STRUCT, BYTE]$$

Let us assume we have some 'obvious' flattening of structures into a non-empty finite sequence of bytes:

$$| \quad \mathit{flatten} : STRUCT \to \mathrm{seq}_1\, BYTE$$

The function is total: all structures can be flattened. Also, all structures have some content: no structure flattens to an empty sequence. The function may also be injective, when there is no overloading of structures, so all structures are represented as different byte streams. (This occurs in the Java Constant Pool case, where the different kinds of structure include a 'tag'. C-style unions, on the other hand, do not include such information, and so their flatten function is not injective, and hence not invertible.)

We can capture both components of the dual representation in a schema.

- *struct* is the structured representation
- *array* is the byte representation
- *index* provides a mapping from each entry in the structured form to the corresponding position in the byte form.

*StructuredArray*1

$$
\begin{array}{l}
struct : \operatorname{seq} STRUCT \\
array : \operatorname{seq} BYTE \\
index : \operatorname{iseq} \mathbb{N}_1 \\
\hline
array = \frown / (struct \fatsemi flatten) \\
\operatorname{dom} index = \operatorname{dom} struct \\
index = \{\ s : \operatorname{seq} STRUCT \mid s \subset struct \bullet \\
\qquad 1 + \# s \mapsto 1 + \#(\frown / (s \fatsemi flatten))\ \}
\end{array}
$$

Because *index* is an injection, it is invertible, and so also provides a mapping from indexes in the byte form back to the corresponding structure. The range of *index* is precisely those array indexes that map back to the beginning of valid structures.

In general, the indexing of the structure and byte streams need not start at one. The byte stream in particular often starts at zero. Let us say that the structures start indexing at i_s and the byte representation at i_a:

$$
i_s, i_a : \mathbb{Z}
$$

We can then write:

StructuredArray

$$
\begin{array}{l}
struct : \operatorname{str} STRUCT \\
array : \operatorname{str} BYTE \\
index : \operatorname{istr} \mathbb{Z} \\
\hline
array = (\ \lambda i : \mathbb{Z} \bullet i + i_a - 1\ ) \fatsemi (\frown / (struct \fatsemi flatten)) \\
\operatorname{dom} index = \operatorname{dom} struct \\
struct \neq \varnothing \Rightarrow \min(\operatorname{dom} struct) = i_s \land \min(\operatorname{dom} array) = i_a \\
index = \{\ s : \operatorname{str} STRUCT \mid s \subset struct \bullet \\
\qquad i_s + \# s \mapsto i_a + \#(\frown / (s \fatsemi flatten))\ \}
\end{array}
$$

Chapter 41

Neural networks

41.1 Introduction

This chapter illustrates a simple use of real numbers, and a use of homogenous binary relations as graphs, or networks.

41.2 A network

Neural networks are used as pattern recognisers and classifiers. They are so called because they are (simplistic) models of the way the brain neurons may work.

We model a neural network as a connected graph of nodes.

$$[NODE]$$

41.2.1 A simple network

A *Net* has the following components.

- *net* captures the relation between the nodes
- *val* is the current activation value of the nodes, represented as a mapping from nodes to their real-valued activations

$$ActVal == NODE \pfun \mathbb{R}$$

- The connections between nodes have a *weight*, describing how much of a node's activation is passed to the connected node, represented as a mapping from pairs of nodes to their real-valued weights

$$Weight == NODE^{2\times} \pfun \mathbb{R}$$

- The *activation* function defines how a node's own activation is related to the inputs it receives.
- Subsets of the nodes are designated as the *in*put nodes, where the pattern to be recognised is input to the network, and *out*put nodes, where the network's response to the pattern is generated.

Net

$$net : NODE \leftrightarrow NODE$$
$$val : ActVal$$
$$weight : Weight$$
$$activation : \mathbb{R} \rightarrow \mathbb{R}$$
$$in, out : \mathbb{P}\, NODE$$

$$\text{dom}\, weight = net \in \text{connected}\, NODE$$
$$\text{dom}\, val = \text{vertex}\, net$$
$$in \cup out \subseteq \text{vertex}\, net$$

The activation function enables thresholding: output from a node occurs only when the inputs reach some threshold. (In practice the function is usually a differentiable sigmoid function.)

41.2.2 A feed forward network

A *feed forward* network has no nodes with outputs feeding back to themselves or earlier nodes.

FeedForwardNet

$$Net$$

$$net \in \text{acyclic}\, NODE$$
$$in = \text{root}\, net^{\sim}$$
$$out = \text{root}\, net$$

In this case the output nodes are ones not connected to further nodes, that is, the roots of the graph (remember our convention that the graph relation points from the leaves to the roots). Similarly, the input nodes are the leaves (or roots of the inverse graph).

41.2.3 A layered network

A *layered* network is a feed forward network where the nodes are arranged in layers, and nodes in a layer are connected only to nodes in the next layer.

LayeredNet

$$
\begin{array}{l}
FeedForwardNet \\
\hline
\exists_1\, s : \operatorname{seq} \mathbb{P}\, NODE \bullet \\
\quad s \operatorname{partition} (\operatorname{vertex} net) \\
\quad \land\ net \subseteq \{ i : 1 \mathinner{..} (\# s - 1) \bullet s\ i \times s\ (i + 1) \}
\end{array}
$$

Equality holds in the *net* predicate when each layer is fully connected to the next.

41.3 Pattern Recognition

A recognition step involves presenting the network with some pattern at its input nodes, and reading off the output it calculates.

Recognise

$$
\begin{array}{l}
\Delta Net \\
in?, out! : ActVal \\
\hline
\operatorname{dom} in? = in \\
val' = in? \cup (\ \lambda\, n : \operatorname{dom} val \setminus in \bullet \\
\qquad \Sigma\ \lambda\, m : \operatorname{predecessors} net\ n \bullet (val \oplus in?) m * weight(m, n)\ ) \\
\Xi Net \setminus (val) \\
out! = out \lhd val'
\end{array}
$$

The value of each node becomes the weighted sum of its inputs, passed through the activation function. The output is then simply the value of the output nodes. No other component of the net changes during recognition.

41.4 Net Training

In order to recognise patterns successfully, a neural network must be *trained*; it must have its weights adjusted to appropriate values.

Train

$$\Delta Net$$
$$train : ActVal^{2\times} \times Weight \rightarrow Weight$$
$$actual?, correct? : ActVal$$

$$\text{dom } actual? = \text{dom } correct? = out$$
$$weight' = train((actual?, correct?), weight)$$
$$\Xi Net \setminus (weight)$$

The inputs consist of the actual output the network provided, and the correct output that it should have provided. The training function uses these and the current weights, to calculate a better set of weights.

Various training functions are used, depending on network topology and error properties. One of the most popular is 'back propagation', which minimises the squared error in a layered network with a monotonic, differentiable activation function, by using a 'hill climbing' algorithm. The details of the algorithm are beyond the scope of this example.

41.5 Further Reading

[Bishop 1995] is a comprehensive textbook on the use of neural networks for pattern recognition. [Machado & Meira 1995] use MooZ, an object oriented extension to Z, to classify the various kinds of neural nets in an inheritance hierarchy.

Chapter 42

Kinship

And so do his sisters, and his cousins, and his aunts!
His sisters, and his cousins,
Whom he reckons up by dozens,
And his aunts!

— W. S. Gilbert, *H.M.S. Pinafore*, 1878

42.1 Introduction

This chapter provides an exploration of the possibilities and nomenclature of kinship, using a Z model. We need not specify how our abstraction maps onto the real world, but the intention is to mirror English kinship language as applied to human beings, and to many animals.

We make heavy use of sequences and relational composition, and the properties of trees and irreflexives orders.

42.2 Ancestors

We start with the set of all individuals

$$[BODY]$$

Everything which follows is equally valid whether $BODY$ is finite or infinite. We divide the set BODY into two sexes in the usual way

$$\begin{array}{|l} female, male : \mathbb{P}\, BODY \\ \hline \langle female, male \rangle \,\text{partition}\, BODY \end{array}$$

so every $BODY$ is either *female* or *male* and no $BODY$ is both.

mother is a function from somebody to their mother, who is female; the relationship is acyclic (nobody is their own mother, or grandmother, and so on). Similarly for *father*, who is male.

A parent is a mother or a father. An ancestor is a parent, or a parent's parent, and so on.

$$
\begin{array}{|l}
mother : BODY \pfun female \\
father : BODY \pfun male \\
parent, ancestor : BODY \rel BODY \\
\hline
parent = mother \cup father \in \text{acyclic}\, BODY \\
parent \in \text{locallyFinite}\, BODY \\
ancestor = parent^+
\end{array}
$$

We require the *parent* relationship to be acyclic. This requirement, forbidding anybody to be their own ancestor, we refer to as the 'axiom of ancestry'. From it we can deduce that the ancestor relationship is an irreflexive order.

$$\vdash ancestor \in \text{irreflexiveOrder}\, BODY$$

We also require *parent* to be locally finite: parents have only a finite number of children.

From the axiom of ancestry, and from the functionality of the *mother* and *father* relations, we can deduce that these relations are *forest*s.

$$\vdash \{mother, father\} \subseteq \text{forest}\, BODY$$

mother and *father* are partial functions, since that is sufficient for what we need to say. To define something as a partial function in Z is to leave open the question as to whether it is total, not to deny it. If *BODY* is finite and non-empty, however, neither *mother* nor *father* can be total: somebody must be parentless (that is, the root of the graph), given the lack of cyclic relationships (law 24.55).

It would be possible to start with the parent-child relationship the other way round, but then we would have to state separately that anybody has only one mother and only one father. As it is we go from parent to child by inverting the *parent* relation.

Mitochondrial DNA is transmitted from mother to child, with no contribution from the father. The mitochondrial DNA in all humans alive today is descended entirely from one woman, dubbed *mitochondrial Eve*, changed only by mutation.

We can use the relationships defined so far to specify this *mitochondrial Eve*, the most recent common female-line ancestor of a population.

$$\begin{aligned} &mitochondrialEve == \lambda\, popn : \mathbb{F}\, BODY \mid \\ &\qquad (\ \exists\, mums : \mathrm{tree}\, BODY \mid mums \subseteq mother \bullet popn \subseteq \mathrm{dom}\, mums\ ) \bullet \\ &\quad \mathrm{lub}\, mother^* popn \end{aligned}$$

For non-trivial populations, Mitochondrial Eve must have had two daughters (if she had only one, that daughter would instead be mitochondrial Eve).

$$\begin{aligned} &popn : \mathbb{F}\, BODY;\ mE : female \mid \\ &\qquad popn \mapsto mE \in mitochondrialEve \\ &\qquad \land\ mE \notin popn \cup mother(\!|\ popn \cap male\ |\!) \\ &\vdash \\ &2 \leq \mathrm{inDegree}(female \lhd mother) mE \end{aligned}$$

42.3 The royal house of Thebes

It is usual to illustrate kinship case-studies from a royal family. We follow that convention, using the royal family of ancient Thebes in Greece (not the ancient Thebes in Egypt, where the pharaohs' family trees were sometimes even more complicated). This gives us some interesting relationships to study, since Jocasta married her own son, Oedipus, by whom she had daughters Antigone and Ismene.

Let $BODY$ be a type containing at least the seven distinct elements

$$\{antigone, creon, haemon, ismene, jocasta, menoeceus, oedipus\}$$

with

$$\begin{aligned} female &= \{antigone, ismene, jocasta\} \\ male &= \{creon, haemon, menoeceus, oedipus\} \end{aligned}$$

The mother and father relationships are

$$\begin{aligned} mother &= \{antigone \mapsto jocasta, ismene \mapsto jocasta, oedipus \mapsto jocasta\} \\ father &= \{antigone \mapsto oedipus, creon \mapsto menoeceus, haemon \mapsto creon, \\ &\qquad ismene \mapsto oedipus, jocasta \mapsto menoeceus\} \end{aligned}$$

that is, *antigone*'s mother is *jocasta*, and so on. These relations obey the required constraints: they are forests (functional acyclic graphs) with female and male ranges respectively.

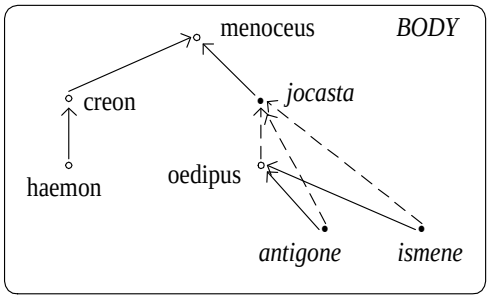


Figure 42.1 The *parent* relation for the royal house of Thebes. The *mother* forest is shown with dashed arrows; the *father* forest is shown with solid arrows. Members of the set *female* are indicated by *slanted* text; members of the set *male* are indicated by roman text.

The *parent* relation is the union of these two (figure 42.1), namely:

$$\begin{aligned} parent = \{&antigone \mapsto jocasta, antigone \mapsto oedipus, creon \mapsto menoeceus,\\ &haemon \mapsto creon, ismene \mapsto jocasta, ismene \mapsto oedipus,\\ &jocasta \mapsto menoeceus, oedipus \mapsto jocasta\} \end{aligned}$$

and we see it is indeed acyclic.

The *ancestor* relation is the irreflexive transitive closure of *parent*, namely:

$$\begin{aligned} ancestor = \{&antigone \mapsto jocasta, antigone \mapsto oedipus, antigone \mapsto menoeceus,\\ &creon \mapsto menoeceus, haemon \mapsto creon, haemon \mapsto menoeceus,\\ &ismene \mapsto jocasta, ismene \mapsto menoeceus, ismene \mapsto oedipus,\\ &jocasta \mapsto menoeceus, oedipus \mapsto jocasta, oedipus \mapsto menoeceus\} \end{aligned}$$

and we see that it is indeed an irreflexive partial order.

42.4 The ancestral line

Ancestors are established by a chain of parenthood. We define the *ancestral line*, a sequence going from the ancestor down through the descendants:

$$line == \text{path}_1\ parent$$

Since the *parent* relation is acyclic (the axiom of ancestry), *line*s are injective paths of ancestors:

$$\vdash line = \text{ipath}_1\ parent$$

We choose to write seq_1 (non-empty, finite sequence) to show that we are not interested in empty lines, but that we do require them to be finite, because we are interested in the *head* and *last* elements of these lines. The axiom of ancestry implies that we can write iseq (injective sequence) for any permissible line, as the ancestral line may contain no duplicates. With inbreeding, however, one body may be the ancestor of another in more than one way; there may be several different lines, not necessarily of the same length, connecting the same pair of bodies.

The lines we can generate from our example are:

$\langle menoeceus, creon, haemon \rangle$
$\langle menoeceus, jocasta, antigone \rangle$
$\langle menoeceus, jocasta, ismene \rangle$
$\langle menoeceus, jocasta, oedipus, antigone \rangle$
$\langle menoeceus, jocasta, oedipus, ismene \rangle$

together with all the infix sequences embedded within these lines.

The ancestor relation is transitive, that is:

$$ancestor \fatsemi ancestor = ancestor$$

We can use the definition of *line* to give an alternative formulation of *ancestor*, in terms of the begining and end body in a line.

$$\vdash ancestor = \{\ l : line \mid 2 \leq \# l \bullet last\ l \mapsto head\ l\ \}$$

Sketch Proof that *ancestor*'s definition could be replaced by the *line* formulation:

The current definition is

$$ancestor = parent^+ = \bigcup \{\ n : \mathbb{N}_+ \bullet parent^n\ \}$$

The definition of relational iteration is in terms of relational composition, which in turn is defined in terms of the existence of a linking element, which becomes the constituent of our line.

We can show by induction on n that there is a line within $parent^n$ whose length is $n + 1$. This allows us to say

$$parent^n = \{\ l : line \mid \# l = n + 1 \bullet \text{last}\, l \mapsto \text{head}\, l\ \}$$

So we express ancestor as

$$\bigcup \{\ n : \mathbb{N}_+ \bullet \{\ l : line \mid \# l = n + 1 \bullet \text{last}\, l \mapsto \text{head}\, l\ \}\ \}$$

which we rewrite as

$$\{\ l : line \mid (\ \exists\, n : \mathbb{N}_+ \bullet \#l = n + 1\ ) \bullet \operatorname{last} l \mapsto \operatorname{head} l\ \}$$

which reduces to the new formulation.

To show that the new formulation is equivalent to the definition, proceed in a similar manner as above.

■

42.5 Matrilineal and patrilineal relations

We can make the more restricted definitions:

$$motherLine == \{\ l : line \mid \operatorname{ran}(\operatorname{front} l) \subseteq female\ \}$$
$$fatherLine == \{\ l : line \mid \operatorname{ran}(\operatorname{front} l) \subseteq male\ \}$$

For example:

$$\langle menoeceus, oedipus, antigone\rangle \in fatherLine$$
$$\langle jocasta, oedipus\rangle \in motherLine$$

Between any pair of bodies there can be at most one mother's line and at most one father's line.

$$a, b : BODY \vdash \#\{\ ml : motherLine \mid \operatorname{head} ml = a \land \operatorname{last} ml = b\ \} \leq 1$$
$$\land\ \#\{\ fl : fatherLine \mid \operatorname{head} fl = a \land \operatorname{last} fl = b\ \} \leq 1$$

We could define matrilineal or patrilineal ancestors, using *motherLine* or *fatherLine* respectively instead of *line* in the definition. This might be useful for describing, for example, mitochondrial DNA inheritance in the matrilineal case, or the transmission of Y chromosomes or British surnames in the patrilineal case. We gave a definition of mitochondrial Eve earlier, using *mother*. An alternative definition, using *motherLine*, is:

MitochondrialEve

$$
\begin{array}{l}
eve : female \\
lines : \mathbb{F}\, motherLine \\
kin : \mathbb{F}\, BODY \\
\hline
\forall\, l : lines \bullet \langle eve \rangle \frown l \in motherLine \\
2 \leq \#\{\ l : lines \bullet \mathrm{first}\, l\ \} \\
kin = \{\ l : lines \bullet \mathrm{last}\, l\ \}
\end{array}
$$

We observe that different human societies differ in the relative importance they attach to the different lines. Some societies stress the father's line to the practical exclusion of the mother's. Conversely there have been, and still are, societies which do not recognise the existence of paternity at all, and where the mother's line is therefore all there is.

Other languages may distinguish matrilineal from patrilineal relations. Swedish, for example, has different words for grandparents on mother's side from those on father's side: *farfar*, 'father's father'; *morfar*, 'mother's father'; *farmor*, 'father's mother'; *mormor*, 'mother's mother'.

Latin distinguises the matrilineal from the patrilineal case for aunts, uncles and cousins. The English word 'aunt' comes from the Latin *amita*, 'father's sister'; 'mother's sister' is *matertera*. The English word 'uncle' comes from the Latin *avunculus*, 'mother's brother'; 'father's brother' is *patruus*.

The English word 'cousin', which means a collateral relative more distant than a brother or sister, comes from the Latin *consobrinus*/*consobrina*, '(first) cousin', Modern English usage is often restricted to first cousin. However, in mediaeval times, the word was often taken to be from the Latin *consanguineus*, 'related by blood', and hence used to refer to any kinsman or kinswoman.

In fact, the Latin legal terminology for cousins is more complicated, distinguishing three (non-disjoint) cases: *fratres patrueles*/*sorores patrueles* for male/female children of two brothers (hence, my father's brother's children); *consobrinus*/*consobrina* for a male/female child of two sisters (hence, my mother's sister's child); and *amitinus*/*amitina* for a male/female child of a father's sibling.

Most of the following theory holds without change for matrilineal or patrilineal relationships. We note some of the differences where they occur.

42.6 Specialisations of the ancestor relation

Grandparents could be defined as fixed compositions of the parent relation, but it gives more flexibility if we make explicit reference to lines.

$$
\begin{array}{|l}
\mathit{greatNthGrandparent} : \mathbb{N} \to \mathit{BODY} \leftrightarrow \mathit{BODY} \\
\hline
\forall\, n : \mathbb{N} \bullet \\
\quad \mathit{greatNthGrandparent}\ n = \{\ l : \mathit{line} \mid \#l = n + 3 \bullet \mathrm{last}\, l \mapsto \mathrm{head}\, l\ \} \\
\end{array}
$$

In the definition of *greatNthGrandparent*, we use a number to count the number of occurrences of the word 'great' in the ordinary English term, from zero upwards. (Latin uses simple prefixes instead of a string of repeated 'great's: 'great' is *pro-*; 'great^2' is *ab-*; 'great^3' is *ad-* (or *at-*); 'great^4' is *tri-* (or *trit-*). Hence '(paternal) great-aunt' is *proamita*, '(maternal) great-great-uncle' is *abavunculus*, 'great-great-great-grandmother' is *atavia*, and 'great-great-great-great-grandson' is *trinepos*.)

Thus we express 'grandparent' as *greatNthGrandparent* 0, 'great-great-great-grandparent' as *greatNthGrandparent* 3, and so on. A similar convention is followed for all relationships which may involve repeated occurrences of the word 'great' such as 'great-aunt'. In all such cases we have the option of declaring the more usual nomenclature by saying, for example:

$$
\begin{aligned}
&\mathit{grandparent} == \mathit{greatNthGrandparent}\ 0 \\
&\mathit{greatGrandparent} == \mathit{greatNthGrandparent}\ 1
\end{aligned}
$$

and so on. In our example

$$
\begin{aligned}
\mathit{grandparent} = \{&\mathit{antigone} \mapsto \mathit{jocasta}, \mathit{antigone} \mapsto \mathit{menoeceus}, \\
&\mathit{haernon} \mapsto \mathit{menoeceus}, \mathit{ismene} \mapsto \mathit{jocasta}, \\
&\mathit{ismene} \mapsto \mathit{menoeceus}, \mathit{oedipus} \mapsto \mathit{menoeceus}\} \\
\mathit{greatGrandparent} = \{&\mathit{antigone} \mapsto \mathit{menoeceus}, \mathit{ismene} \mapsto \mathit{menoeceus}\}
\end{aligned}
$$

Grandmothers and grandfathers, and so on, can be described by range restricting the above relations, or by including qualifications such as:

$$
\begin{aligned}
&\mathrm{head}\, l \in \mathit{female} \\
&\mathrm{head}\, l \in \mathit{male}
\end{aligned}
$$

in the definitions.

The transitivity of the ancestor relation now takes the form of simple theorems such as:

$$
\begin{aligned}
n : \mathbb{N} \vdash\ & parent \fatsemi great\mathit{NthGrandparent}\ n \\
& = great\mathit{NthGrandparent}\ n \fatsemi parent \\
& = great\mathit{NthGrandparent}(n+1) \\
m, n : \mathbb{N} \vdash\ & great\mathit{NthGrandparent}\ m \fatsemi great\mathit{NthGrandparent}\ n \\
& = great\mathit{NthGrandparent}(m+n+2)
\end{aligned}
$$

42.7 Descendants

Children, descendants and grandchildren can be defined as the inverse of parents, ancestors and grandparents.

$$
\begin{array}{|l}
child, descendant : BODY \leftrightarrow BODY \\
great\mathit{NthGrandchild} : \mathbb{N} \to BODY \leftrightarrow BODY \\
\hline
child = parent^{\sim} \\
descendant = ancestor^{\sim} \\
\forall\, n : \mathbb{N} \bullet great\mathit{NthGrandchild}\ n = (great\mathit{NthGrandparent}\ n)^{\sim}
\end{array}
$$

In our example

$$
\begin{aligned}
child = \{ & creon \mapsto haemon, jocasta \mapsto antigone, jocasta \mapsto ismene, \\
& jocasta \mapsto oedipus, oedipus \mapsto antigone, oedipus \mapsto ismene, \\
& menoeceus \mapsto creon, menoeceus \mapsto jocasta\}
\end{aligned}
$$

We can now define daughters and sons, granddaughters and grandsons, and so on, by range restriction, or other means.

There are similar transitivity properties for *descendants* as for *ancestors*.

42.8 Collateral relations

We want to define general 'collateral' relationships. A collateral relationship is one of common descent, but in different lines. So siblings are collaterally related, as are aunt and nephew, but father and daughter are not, because the father's line is an infix of the daughter's line.

So two bodies are collaterally related if they have a common ancestor, but by different lines. Their relationship in general is defined by their *nearest* common ancestor: two bodies are siblings if they have common parents, and hence common grandparents; they are cousins if they have common grandparents but different parents.

42.8.1 Disjoint lines

Two bodies are related if they are at the end of two lines that share a head body, but have no other bodies in common.

CollateralAncestor

$$hd : BODY$$
$$l_1, l_2 : line$$
$$kin : BODY \times BODY$$

$$\{\langle hd \rangle \frown l_1, \langle hd \rangle \frown l_2\} \subseteq line$$
$$\text{disjoint}\langle \text{ran}\, l_1, \text{ran}\, l_2 \rangle$$
$$kin = (\text{last}\, l_1, \text{last}\, l_2)$$

The disjointness predicate in *CollateralAncestor* prevents spurious recognition of more distant relationships between two bodies where the real relationship is closer; we want a sensible nomenclature even when the population is highly inbred. It emerges as we pursue the issue that this predicate is perhaps too weak, but it will do for the time being. In any case we are not prevented from recognising multiple collateral links between the same pair of bodies.

hd is prevented from occurring in l_1 or l_2, from the definition of *line*.

Below we define common named relationships. When we define the set of *kin* comprising the relationship *aunt* say, then the existence of the kin pair (a, b) in the relationship means that a is the aunt of b.

42.8.2 Common ancestors

There may be a common female ancestor, a common male ancestor, or both. We define schemas expressing the existence of a common female or male ancestor:

$$FemaleAncestor == [\ CollateralAncestor \mid hd \in female\ ]$$
$$MaleAncestor == [\ CollateralAncestor \mid hd \in male\ ]$$

If we were developing the theory of matrilineal, or of patrilineal relationships, we should use the appropriate one of these two, but using *motherLine* or *fatherLine* instead of *line* within the schemas. For our sexless theory, however, we put them together using schema disjunction. The resultant schema describes a situation where the two lines, l_1 and l_2, have at their heads a common female ancestor, a common male ancestor, or both.

We can alternatively put these schemas together using schema conjunction. The resultant schema describes a situation where the two lines, l_1 and l_2, have at their heads both a common female ancestor and a common male ancestor (for example, the same mother, and the same father).

$$AncestorPair == FemaleAncestor \land MaleAncestor$$

To express the situation where there is a female common ancestor or a male common ancestor, but not both, at the heads of the same lines (for example, the same mother, but a different father), we use a conjunction with a schema negation.

$$SingleAncestor == CollateralAncestor \land \neg AncestorPair$$

42.9 Siblings

Siblings (brothers and sisters) of various sorts are one's parents' children, excluding oneself.

Sibling

$$CommonAncestor$$

$$\#l_1 = 1 = \#l_2$$

We define some variants: *sib* for pairs with a common mother or father, *maternalSib* for pairs with a common mother, *paternalSib* for pairs with a common father, *fullSib* for pairs with a common mother and father, *halfSib* for pairs with a common mother or father but not both.

$$
\begin{array}{|l}
sib, maternalSib, paternalSib, fullSib, halfSib : BODY \leftrightarrow BODY \\
\hline
sib = \{\ Sibling \bullet kin\ \} \\
maternalSib = \{\ FemaleAncestor;\ Sibling \bullet kin\ \} \\
paternalSib = \{\ MaleAncestor;\ Sibling \bullet kin\ \} \\
fullSib = \{\ AncestorPair;\ Sibling \bullet kin\ \} \\
halfSib = \{\ SingleAncestor;\ Sibling \bullet kin\ \}
\end{array}
$$

All sibling relations are symmetric (I am always my sibling's sibling), but not all are transitive (if I have a paternal half sibling, they may have a maternal half sibling to whom I am not necessarily related).

We can turn the definitions of siblings into ones for sisters and brothers by modification of *Sibling* or of the definitions, adding the predicate

$$last\ l_2 \in female$$
$$last\ l_2 \in male$$

or by the appropriate range restrictions.

The definitions of sibling and other collateral relationships are made more complex than they might be by the way in which our culture and our language recognise monogamy as the norm. Were that not so, we should have simple words for half-sister and half-brother, and being a full sibling would be regarded as a coincidental case where two bodies were related twice over.

The word 'uterine' is sometimes used for the qualification we have called 'maternal'. The old-fashioned phrases 'sister german' and 'brother german' mean *fullSib*. (In this sense, 'german', a variant spelling of 'germane', comes from the Latin *germanus*, 'having the same parents; genuine, real'.) Qualifications such as 'half' and 'full' are not used in normal English for relations other than siblings.

42.10 Aunt, uncle, niece, nephew, cousin

We define the remaining collateral relations using *CollateralAncestor*, which is adequate for modelling English kinship language. For other purposes we might wish to define separate categories of collateral relations using *AncestorPair* and *SingleAncestor*.

The aunt-or-uncle, and niece-or-nephew relationships (figure 42.2) are defined as:

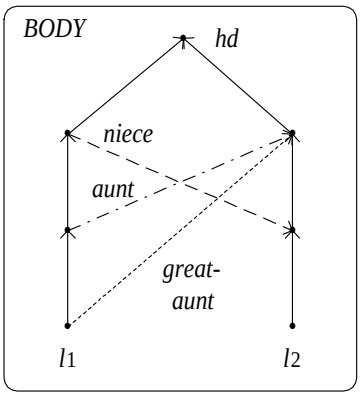


Figure 42.2 Great-aunt, aunt and niece relationships are shown as dashed lines. The solid lines indicate the mother relation.

$$
\begin{array}{|l}
\mathit{greatNthAuntUncle} : \mathbb{N} \to \mathit{BODY} \leftrightarrow \mathit{BODY} \\
\mathit{greatNthNieceNephew} : \mathbb{N} \to \mathit{BODY} \leftrightarrow \mathit{BODY} \\
\hline
\forall\, n : \mathbb{N} \bullet \\
\quad \mathit{greatNthAuntUncle}\ n = \\
\qquad \{\ \mathit{CollateralAncestor} \mid \#l_1 = n + 2 \land \#l_2 = 1 \bullet \mathit{kin}\ \} \\
\quad \land\ \mathit{greatNthNieceNephew}\ n = \\
\qquad \{\ \mathit{CollateralAncestor} \mid \#l_1 = 1 \land \#l_2 = n + 2 \bullet \mathit{kin}\ \}
\end{array}
$$

We can specialise into aunts, uncles, nieces and nephews separately, by range restriction or by adding the sex qualification into the predicate.

The aunt-or-uncle relation is the inverse of the niece-or-nephew relation:

$$n : \mathbb{N} \vdash \mathit{greatNthAuntUncle}\ n = (\mathit{greatNthNieceNephew}\ n)^{\sim}$$

The old phrase ‘cousin german’ means first cousin. The cousin relationships (figure 42.3) are defined as:

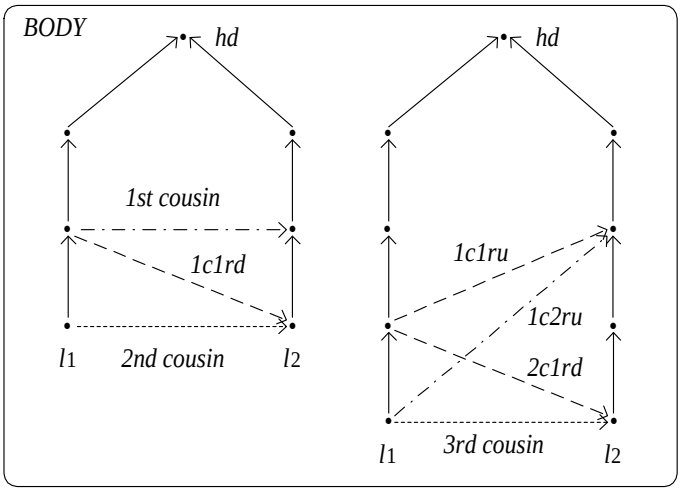


Figure 42.3 Cousin relationships are shown as dashed lines. $1c1rd$: 1st cousin once removed downwards; $1c1ru$: 1st cousin once removed upwards; $1c2ru$: 1st cousin twice removed upwards; $2c1rd$: 2nd cousin once removed downwards. The solid lines indicate the parent relation.

$$
\begin{array}{|l}
nthCousin : \mathbb{N}_+ \to BODY \leftrightarrow BODY \\
nthCousinMTimesRemovedUpwards : \mathbb{N}_+ \to \mathbb{N}_+ \to BODY \leftrightarrow BODY \\
nthCousinMTimesRemovedDownwards : \mathbb{N}_+ \to \mathbb{N}_+ \to BODY \leftrightarrow BODY \\
\hline
\forall\, n : \mathbb{N}_+ \bullet nthCousin\ n = \\
\qquad \{\ CollateralAncestor \mid \#l_1 = n + 1 \land \#l_2 = n + 1 \bullet kin\ \} \\
\forall\, m, n : \mathbb{N}_+ \bullet \\
\qquad nthCousinMTimesRemovedUpwards\ n\ m = \\
\qquad\qquad \{\ CollateralAncestor \mid \#l_1 = m + n + 1 \land \#l_2 = n + 1 \bullet kin\ \} \\
\qquad nthCousinMTimesRemovedDownwards\ n\ m = \\
\qquad\qquad \{\ CollateralAncestor \mid \#l_1 = n + 1 \land \#l_2 = m + n + 1 \bullet kin\ \}
\end{array}
$$

Cousinship upwards is the inverse of cousinship downwards:

$$
\begin{aligned}
m, n : \mathbb{N}_+ \vdash\ & nthCousinMTimesRemovedUpwards\ n\ m \\
& = (nthCousinMTimesRemovedDownwards\ n\ m)^\sim
\end{aligned}
$$

Notice that English has no sex-free term for niece-or-nephew, or for aunt-or-uncle. On the other hand, it has no term to distinguish male cousins from female cousins.

If we were designing relationship terminology from scratch, instead of formalising exisiting terminology, we could name *all* the collateral relationships in a uniform manner, by redefining sibling, aunt-or-uncle and niece-or-nephew as 'zeroth cousins':

- sibling becomes zeroth cousin
- aunt-or-uncle becomes zeroth cousin once removed upwards
- great-n-aunt-or-uncle becomes zeroth cousin n times removed upwards
- niece-or-nephew becomes zeroth cousin once removed downwards
- great-n-niece-or-nephew becomes zeroth cousin n times removed downwards

42.11 General theorem of blood relationship

We define a general category for all the collateral relations:

$$\begin{array}{|l} collateral : BODY \leftrightarrow BODY \\ \hline collateral = \{\ CollateralAncestor \bullet kin\ \} \end{array}$$

We have given names to various special kinds of collateral relationships, which partition the complete set into 'sisters and cousins and aunts' (along with the male variants and the inverses):

$$\begin{array}{l} \vdash collateral = \\ \quad sib \\ \quad \cup \{\ n : \mathbb{N} \bullet greatNthAuntUncle\ n \cup greatNthNieceNephew\ n\ \} \\ \quad \cup \{\ n : \mathbb{N}_+ \bullet nthCousin\ n\ \} \\ \quad \cup \{\ m, n : \mathbb{N}_+ \bullet nthCousinMTimesRemovedUpwards\ n\ m \\ \qquad\qquad \cup\ nthCousinMTimesRemovedDownwards\ n\ m\ \} \end{array}$$

We now come to the general theorem of blood-relationship, namely:

$$\begin{array}{l} \vdash collateral \\ \quad \subseteq ancestor \fatsemi descendant \\ \qquad \subseteq \mathrm{id}\ BODY \cup ancestor \cup descendant \cup collateral \end{array}$$

Sketch Proof of the theorem of blood relationship:

The first half of the theorem follows straight from the definition. The second half is shown as follows:

Take an arbitrary pair from *ancestor* ⨾ *descendant*. Relational composition implies the existence of a linking member, which must be female or male. We detach this member from the lines of the ancestor and descendant, whose length is at least 2, and call the beheaded lines l_1 and l_2, with length at least 1. This gives us the structure of *Collateral* except perhaps for its disjointness predicate.

Consider each domain-range pair in the light of this predicate.

Where the predicate is true, we have a member of the set *collateral*.

Otherwise the intersection set is non-empty. Choose the $BODY$, b say, from this set with highest index in l_1. This must be the same $BODY$ as has highest index in l_2, since otherwise we can show a violation of the axiom of ancestry.

If the chosen $b = \text{last}\, l_1 = \text{last}\, l_2$, then it is equal to both elements of our kin pair, and we have a member of $\text{id}\, BODY$.

If the chosen $b = \text{last}\, l_1 \neq \text{last}\, l_2$, then it is the same as the first member of our kin pair. There is a segment of line from the occurrence of this $BODY$ in l_2 to its end, and we have a member of the set *descendant*. (And similarly the other way around for *ancestor*.)

If the chosen $b \neq \text{last}\, l_1 \wedge b \neq \text{last}\, l_2$, consider it as the common element in *CollateralAncestor*, and the segments of l_1 and l_2 beyond its occurrence in them as the new values of l_1 and l_2;. All the conditions of *Collateral* are now met, including the disjointness of the ranges, and we have a member of the set *collateral*.

■

42.12 Antigone's relations

42.12.1 Antigone's grandparents' grandchildren

Antigone's grandparents' grandchildren might be expected to be her cousins, or her siblings.

Antigone's grandparents are $\{jocasta, menoeceus\}$ and their grandchildren are:

- *antigone*: herself, so here the kin pair is a member of $\text{id}\, BODY$.
- *haemon*: who is related through the l_1 line and the l_2 line, which are disjoint, so Haemon is Antigone's first cousin.
- *ismene*, who is related twice over. In the first case, the l_1 line goes up through

$\langle menoeceus, jocasta, antigone\rangle$ and back down through $\langle menoeceus, jocasta, ismene\rangle$, which lines intersect in *jocasta*, so we ignore *menoeceus*, take *jocasta* as the common head, and use the lines which are now of length 1.
In the second case the l_1 line goes up through $\langle jocasta, oedipus, antigone\rangle$ and back down through $\langle jocasta, oedipus, ismene\rangle$, so we similarly take *oedipus* as the common head and have lines of length 1.
We observe that the two pairs of lines are the same, so we have an *AncestorPair*, and thus establish that Antigone and Ismene are full sisters (with mother Jocasta and father Oedipus).

- *oedipus*. The lines from *antigone* to *oedipus* intersect in *jocasta*, so Oedipus is also Antigone's half-brother.

42.12.2 Antigone's great-grandparents' great-grandchildren

Antigone's great-grandparents' great-grandchildren might be expected to be her cousins (suitably removed), or her siblings, but some relationships turn out to be rather more complicated.

We extend the lines to suppose that Menoeceus had a parent, thus:

$\langle unknown, menoeceus, creon, haemon\rangle$
$\langle unknown, menoeceus, jocasta, antigone\rangle$
$\langle unknown, menoeceus, jocasta, ismene\rangle$
$\langle unknown, menoeceus, jocasta, oedipus, antigone\rangle$
$\langle unknown, menoeceus, jocasta, oedipus, ismene\rangle$

Antigone's great-grandparents are $\{unknown, menoeceus\}$, and their great-grandchildren are:

- *antigone* herself
- *creon*: the lines to and from *creon* intersect in *menoeceus*, so we shorten them by 1 and establish that Creon is Antigone's uncle.
- *haemon*: the lines to and from *haemon* are disjoint, so Haemon is Antigone's cousin once removed. (Note that these lines also establish that Creon is Antigone's great-uncle).
- *ismene*: the lines to and from *ismene* intersect in *jocasta*, so we shorten by 1 and see that Ismene is Antigone's aunt.
- *jocasta*: the lines to and from *jocasta* intersect in both *menoeceus* and *jocasta*, reducing to the fact that Jocasta is Antigone's mother.

- *oedipus*: the lines to and from *oedipus* intersect in both *jocasta* and *oedipus*, reducing to the fact that Oedipus is Antigone's father.

42.12.3 Summary

Summarising Antigone's relations, we end up with:

- Antigone is herself
- Creon is her uncle and great-uncle
- Haemon is her first cousin and first cousin once removed
- Ismene is her full sister, niece, and aunt
- Jocasta is her mother and grandmother
- Menoeceus is her grandfather and great-grandfather
- Oedipus is her father and half-brother

As readers of Sophocles will be aware, Haemon is also Antigone's fiancé.

42.13 Monotonic functions on sets

We might take the view that Ismene, being full sister to Antigone, should not also be considered to be her aunt and niece. It is possible, by modifying the predicate in *CollateralAncestor*, to arrive at a formulation where Ismene is Antigone's sister only. (We leave further investigation of possible modifications to the definitions as an exercise to the reader.) We should be wary of such a redefinition, however, because it would prevent our collateral relationships from being *monotonic*.

Our relationships are defined with respect to the information in the type *BODY* and the two base relations *mother* and *father*. If we add more information into those sets (for example, by adding in Menoeceus's parents), there are two possibilities: either this new information causes us to *contradict* any conclusions about relationships which we have made, or it simply adds new conclusions.

We call our relationships *monotonic* if adding any new information does not cause us to contradict any conclusions. So if we change to new base sets $mother'$ and $father'$, with

$$mother \subseteq mother' \land father \subseteq father'$$

and if *kin* stands for any of the kinship relations we have defined above, then *kin*

is monotonic if we also have

$$kin \subseteq kin'$$

It is clear that most of our relations are indeed monotonic. If we have established that Jocasta is Antigone's grandmother, no extra information about who is parent to whom can alter that fact.

An example of a relation which would not be monotonic in this sense would be *unrelated*. Any statement that two bodies are unrelated is always at risk from the production of new evidence. In fact *unrelated* is *anti*-monotonic, always safe against the deletion of evidence, or the removal of members from the base relations *mother* and *father*.

The only definitions we have given above which are not monotonic are *Single-Ancestor*, and thus also *halfSib*. This corresponds to the fact that if two bodies have a parent in common, and we have no information about the other parent, we might provisionally declare the bodies to be half-sibs. If they then turned out to have the other parent in common as well, we should have to change our minds.

Returning now to the 'Ismene as aunt to Antigone' question, it is apparent that we might know only enough to assert that they were aunt and niece, either way round. If the discovery of further information (that they are sisters) caused us to rescind this decision, we should have lost monotonicity. Being monotonic may be a desirable quality, and tends to lead to simpler theories, but it is not essential. In this case it would probably accord better with commonsense language to have a definition of collateral relationships that is not monotonic. (The details are left as an exercise for the reader.)

42.14 Other compositions

42.14.1 Composition of descendant with ancestor

If we consider the relational composition

$$descendant \fatsemi ancestor$$

we build up relationships describing the ancestors of our descendents. We encounter the striking fact that in English, and other languages, there are no words to describe these relationships. Even the relation 'other parent of my child' lacks a single name ('spouse' is the correct term only in a totally monogamous society, of which there are no examples), as does 'other grandparents of my grandchildren', a relationship usually therefore described in very much those words.

42.14.2 Composition with collateral

The other compositions which are guaranteed to yield something are:

$$\textit{ancestor} \fatsemi \textit{collateral}$$
$$\textit{collateral} \fatsemi \textit{descendant}$$
$$\textit{ancestor} \fatsemi \textit{collateral} \fatsemi \textit{descendant}$$

As with $\textit{ancestor} \fatsemi \textit{descendant}$, however, the resultant relationship is not necessarily the obvious one: it might be something closer.

All remaining compositions, such as

$$\textit{collateral} \fatsemi \textit{collateral}$$
$$\textit{collateral} \fatsemi \textit{ancestor}$$
$$\textit{descendant} \fatsemi \textit{collateral}$$

in general yield nothing which is recognised by our kinship vocabulary.

42.15 Enriching the model

42.15.1 Including birth order

In a more powerful model we might wish to rank the children of any parent in order of age, which we would need in order to encompass things like the inheritance of titles in modern Britain, and the inheritance of property in earlier days. That could be done by defining injections between each parent and the sequence of children in age order.

Some languages would require such an enriched model; for example, Mandarin Chinese distinguishes elder and younger siblings: *ko-ko*, 'elder brother' and *ti-ti*, 'younger brother'. In fact, an even richer model would be required for Chinese, which also distinguishes elder and younger cousins: *t'ang-chieh*, 'father's brother's daughter, older than oneself' and *t'ang-mei*, 'father's brother's daughter, younger than oneself'.

42.15.2 Including relationship by marriage

We have considered only blood relationship above. We could consider relationship by marriage, too. Then we would need to extend our model to capture who was

married to whom, and define 'in-law' relations. Again, the complexity of modelling depends on the detail that needs to be captured. For example, modern English uses 'mother-in-law' to denote just 'spouse's mother', while Mandarin Chinese distinguishes *yueh-mu*, 'wife's mother' from *p'o-p'o*, 'husband's mother'.

If we allow remarriage, we also need 'step-' relations; if we allow divorce, we also need 'ex-' relations.

A new class of confusing relationships can occur in this enriched model. For example, if a mother and daughter marry a son and father respectively, they are each their own step-grandparent.

42.15.3 Relaxing the axiom of ancestry

Recall that the 'axiom of ancestry', the lack of cycles in the ancestor relation, is used to ensure that nobody is their own ancestor.

In what is possibly the ultimate in time-travel stories, '–All You Zombies–' by Robert A. Heinlein, it turns out that the main character is not only their own mother, but also their own father. This amazing feat requires not only a sex change, but also judicious use of a time machine.

If we wanted to model such science-fictional relationships, we would need to make two changes to our model: the *mother* and *father* relations would not be restricted to being acyclic, and *male* and *female* could not be fixed sets given in an axiomatic definition, but would be modelled by schemas, with a corresponding *ChangeSex* operation.

The resulting family relationships would become even more entangled than poor Antigone's!

42.16 Further reading

For a discussion of the various kinship relationships distinguished in various cultures, see, for example, [Fox 1967].

[Mommsen *et al.* 1985, section 38.10] is a primary source for Latin kinship terminology. [Bradley 1991] is a less complete, but more readable, summary.

[Hervey 1979] describes 140 Mandarin Chinese kinship terms, ranging from *mu-ch'in*, 'mother', to *piao-po-mu*, 'wife of son of paternal grandfather's sister or paternal grandmother's sibling, when son is older than one's own father'.

English kinship language has little convenient terminology to refer to 'step-' and 'ex-' relationships. For an irreverant discussion of how the language needs to be extended (in order to refer, for example, to one's ex-husband's current wife's ex-husbands' children) in a society of multiple divorces and remarriages, see [Kaye 1986]. Even without the complication of divorce, it is still the case that grandparents might want to refer to their late son's widow's (new) husband's children by his (deceased) previous wife.

The time travel story '–All You Zombies–' can be found in [Heinlein 1959].

Chapter 43

Chess

43.1 Introduction

We give a specification of the rules of the game of chess [Kazic *et al.* 1985]. Chess serves as a good example because it is a real problem, already well understood, but not trivial. We can concentrate on describing it accurately without being distracted by design issues, as its design has already taken place. We give a complete definition of the logical game, abstracted from its physical representation and from the rules governing the conduct of the players.

43.2 The board

The chessboard is an eight by eight array of squares. The logical game can ignore the fact that the squares are alternately black and white, which is for the sole purpose of assisting the human player in the visual recognition of positions. We represent each square by its two coordinates, row number and file number respectively, so a particular square could be represented by $(1, 1)$ or $(5, 8)$, for example. The set of all squares is the Cartesian product of the row numbers, 1 to 8 inclusive, with the file numbers, also 1 to 8 inclusive. So we define

$$square == 1 \,..\, 8 \times 1 \,..\, 8$$

In conventional chess terminology, rows are fixed in the board, numbering from white's first rank, while ranks, which we introduce below, are dependent on whose turn it is. Files are fixed in the board, numbering from the queen's side. We define selector functions to give the components of the square:

$$row, file : square \rightarrow 1 \,..\, 8$$

$$\forall\, s : square \bullet s = (row\ s, file\ s)$$

We have numbered the files from 1 to 8 for the convenience of this model. In fact the systems in general use are either to use the letters A to H, the algebraic system as it is known, or to identify each file by the piece that appears on it in the initial position, described below. It is an elementary exercise to formalise those descriptions, but they are not required for this specification. Note also that ranks are used in preference to rows. We define ranks below.

43.3 The chessmen

There are exactly six sorts of chessman, distinct from each other and everything else.

$$MAN ::= pawn \mid rook \mid knight \mid bishop \mid queen \mid king$$

A "piece" is any chessman except a pawn, but we need not formalise that.

Chessmen come in two colours, *black* and *white*. A chessman not on the board plays no part in the logical game, and therefore we only need information about a chessman's colour when considering it as an occupant of a square on the board. It is convenient at the same time to formalise the fact that a square may be unoccupied, which leads us to the following definition:

$$OCCUPANT ::= empty \mid black\langle\langle MAN\rangle\rangle \mid white\langle\langle MAN\rangle\rangle$$

This defines occupant as a set with thirteen members, one called *empty* and six each corresponding to the black and white chessmen. *black* and *white* are functions from MAN to $OCCUPANT$. We use them as the definition of the teams:

$$team == \{black, white\}$$

team is the set containing the injections *black* and *white*.

43.4 Board positions

We define a board configuration as a total function from a square to its occupant:

$$position == square \to OCCUPANT$$

A *playPosition* is a *position* together with the information as to whose turn it is to move next:

$$playPosition == team \times position$$

We define selector functions for the play position:

$$\begin{array}{|l} turn : playPosition \rightarrow team \\ board : playPosition \rightarrow position \\ \hline \forall\, pp : playPosition \bullet pp = (turn\ pp, board\ pp) \end{array}$$

A game is a non-empty sequence of *playPosition*s with alternation of play.

$$game == \{\ g : \mathrm{seq}_1\ playPosition \mid \forall\, r, s : \mathrm{dom}\ g \mid s = r + 1 \bullet turn(g\ r) \neq turn(g\ s)\ \}$$

43.5 The starting position

We define the initial position, and thus the initial game.

$$\begin{array}{|l} initGame : game \\ initPos : position \\ backrow == \langle rook, knight, bishop, queen, king, bishop, knight, rook \rangle \\ \hline \forall\, n : 1 \mathinner{..} 8 \bullet initPos(1, n) = white(backrow\ n) \\ \quad \wedge\ initPos(2, n) = white\ pawn \\ \quad \wedge\ (\ \forall\, m : 3 \mathinner{..} 6 \bullet initPos(m, n) = empty\ ) \\ \quad \wedge\ initPos(7, n) = black\ pawn \\ \quad \wedge\ initPos(8, n) = black(backrow\ n) \\ initGame = \langle (white, initPos) \rangle \end{array}$$

backrow is equal to an explicit sequence of chessmen. We spread it along *row* 1 in its white form, and *row* 8 in its black. Rows 2 and 7 are filled with pawns, and the middle four rows are empty.

The initial game is defined to be the sequence comprising the single *playPosition* that has the initial position and *white* to move.

43.6 Moves in general

Now we specify the valid moves. We express this using schemas to describe and constrain particular aspects of the game.

We detach the most recent board position, *pos*, from the end of the game sequence. We declare another board position, pos', which will be appended to the game sequence. We also sort out whose move it is.

Player

$$g, g' : game$$
$$pos, pos' : position$$
$$player, opponent : team$$
$$(player, pos) = \text{last } g$$
$$g' = g \frown \langle (opponent, pos') \rangle$$

g represents the current game sequence, the *last* element of which is the current player and board position *pos*. Then next board position pos' is appended to the game sequence to give the new game sequence g'. The declaration $g' : game$ allows us to infer properties such as $opponent \neq player$

Let us give ourselves the capability to talk in terms of *row*s or of *rank*s, whichever is more convenient.

Ranks

$$Player$$
$$rank : square \rightarrow 1 \,..\, 8$$
$$player = black \Rightarrow (\ \forall s : square \bullet rank\ s + row\ s = 9\ )$$
$$player = white \Rightarrow (\ \forall s : square \bullet rank\ s = row\ s\ )$$

In general, a move involves a chessman that starts on a particular square, the *from* square, which is moved to a different square, the *to* square. To analyse the geometry of the move, we split each dimension (along the *row* and *file*) of the move into a direction and a distance. The direction is 0 (no movement in that dimension), -1 (a move decreasing the *row* or *file* or *rank* number) or $+1$ (a move increasing the *row* or *file* or *rank* number), and always defined. The distance, the number of squares moved in the direction, is strictly positive, but irrelevant where the direction is 0. *rankdir* and *rowdir* are of the same magnitude but may be of opposite sign.

Target

$$
\begin{array}{l}
Ranks \\
from, to : square \\
rdist, fdist : 1 \mathinner{..} 7 \\
rowdir, rankdir, filedir : \{-1, 0, 1\} \\
\hline
from \neq to \\
file\ to = file\ from + filedir * fdist \\
row\ to = row\ from + rowdir * rdist \\
rank\ to = rank\ from + rankdir * rdist
\end{array}
$$

We now identify the chessman that is to move. Typical moves require that the *to* square is either occupied by an opposing chessman, which is "taken", or is empty. Taking a chessman is fairly undramatic in this model. It occurs simply by producing a new board position in which the taken chessman does not appear. In general the new position is the same as the old position except with the *from* square empty and the *to* square occupied by the moved chessman. Castling and most pawn moves are exceptions to this, and therefore do not use this next schema.

Taker

$$
\begin{array}{l}
Target \\
m : MAN \\
\hline
pos\ from = player\ m \\
pos\ to \notin \operatorname{ran} player \\
pos' = pos \oplus \{(from, empty), (to, player\ m)\}
\end{array}
$$

Clear constrains a move to be vertical, horizontal or diagonal, and the intervening space to be empty. The knight's move therefore does not use it. (Recall that for horizontal move that $rowdir = 0$, so the value of *rdist* is irrelevant; similarly for vertical moves and *fdist*.)

Clear

$$
\begin{array}{l}
Target \\
dist : 1 \mathinner{..} 7 \\
\hline
dist = rdist = fdist \\
\forall\, n : 1 \mathinner{..} dist - 1 \bullet \\
\quad pos(row\ from + rowdir * n, file\ from + filedir * n) = empty
\end{array}
$$

43.7 Moves in particular

We now deal with the moves appropriate to each specific kind of chessman.

Rooks can move vertically or horizontally, any distance. Diagonal movement is disallowed by the predicate $0 \in \{rowdir, filedir\}$.

RookMover

$$
\begin{array}{l}
Taker \\
Clear \\
\hline
m = rook \\
0 \in \{rowdir, filedir\}
\end{array}
$$

Knights move one square vertically and two horizontally, or *vice versa*.

KnightMover

$$
\begin{array}{l}
Taker \\
\hline
m = knight \\
0 \notin \{rowdir, filedir\} \\
\{fdist, rdist\} = \{1, 2\}
\end{array}
$$

Bishops move diagonally, any distance. The predicate $0 \notin \{rowdir, filedir\}$ ensures diagonal movement.

BishopMover

$$
\begin{array}{l}
Taker \\
Clear \\
\hline
m = bishop \\
0 \notin \{rowdir, filedir\}
\end{array}
$$

The queen moves vertically, horizontally or diagonally, any distance.

QueenMover

$$
\begin{array}{l}
Taker \\
Clear \\
\hline
m = queen
\end{array}
$$

The king moves vertically, horizontally or diagonally, to an adjacent square.

KingMover

$$
\begin{array}{l}
Taker \\
Clear \\
\hline
m = king \\
dist = 1
\end{array}
$$

A pawn may moves one square forwards onto an empty square, or one square diagonally forwards to take. If a pawn reaches its eighth rank by either method it is "promoted" to a non-king piece. m' is the possibly promoted pawn, declared as being a member of the set MAN excluding the element $king$.

MainPawnMover

$$
\begin{array}{l}
Target \\
Clear \\
m' : MAN \setminus \{king\} \\
\hline
pos\ from = player\ pawn \\
filedir = 0 \Leftrightarrow pos\ to = empty \\
rankdir = 1 \\
dist = 1 \\
m' = pawn \Leftrightarrow rank\ to \neq 8 \\
pos' = pos \oplus \{(from, empty), (to, player\ m')\}
\end{array}
$$

The usual realisation of the logical game, with chessmen that are picked up and moved about a flat board, might suggest that there is some constraint on the total number of occurrences of any kind of chessman to the number physically provided, such as the number of each kind required for the initial position. There is no such constraint. It is theoretically possible to have nine queens of the same colour on the board, if all eight pawns have been promoted.

A pawn may move forward two squares on its first move, provided it is not taking.

FirstPawnMover

$$
\begin{array}{l}
Taker \\
Clear \\
\hline
m = pawn \\
filedir = 0 \\
pos\ to = empty \\
rank\ from = 2 \\
rank\ to = 4
\end{array}
$$

The *enPassant* move applies directly after the double move option for a pawn's first move is taken. If that original pawn had instead moved forward only one space to a square where it could have been taken by an opposing pawn, then the other player may as their immediately following move take that pawn, themselves moving as if the original pawn had moved only the one space.

EnPassant

$Target$
$victim : square$

$pos\ from = player\ pawn$
$rank\ from = 5$
$rank\ to = 6$
$filedir \neq 0$
$rankdir = 1$
$fdist = 1$
$file\ to = file\ victim$
$(front\ g, pos, victim) \in \{FirstPawnMover \bullet (g, pos', to)\}$
$pos' = pos \oplus \{(victim, empty), (to, player\ pawn), (from, empty)\}$

We have defined all moves except castling. Castling requires us to define what it is to be in check, and we can do this now, since castling itself can never directly threaten a king, and so does not affect the definition of check. First we disjoin the moves defined so far:

$$\begin{aligned} PlainMover == {} & RookMover \lor KnightMover \lor BishopMover \\ & \lor QueenMover \lor KingMover \\ & \lor MainPawnMover \lor FirstPawnMover \lor EnPassant \end{aligned}$$

A player is in check if the king could be taken (hence is not on the board) on the next move. We define this formally as a set of *playPosition*s with the vulnerable player as the first of the pair.

$$check == \{PlainMover \mid opponent\ king \notin \operatorname{ran} pos' \bullet (opponent, pos)\}$$

Castler

$$
\begin{array}{l}
Clear \\
k, r : square \\
\hline
rank\ from = rank\ to = rank\ k = rank\ r = 1 \\
file\ from = 5 \\
file\ to = 1 \land file\ k = 3 \land file\ r = 4 \\
\lor\ file\ to = 8 \land file\ k = 7 \land file\ r = 6 \\
\forall\, pp : \operatorname{ran} g \bullet (board\ pp) from = player\ king \\
\qquad \land\ (board\ pp) to = player\ rook \\
(player, pos) \notin check \\
(player, pos \oplus \{(from, empty), (r, player\ king)\}) \notin check \\
pos' = pos \oplus \{(from, empty), (r, player\ rook), \\
\qquad (k, player\ king), (to, empty)\}
\end{array}
$$

That is:

- The whole move takes place on the player's first rank
- Castling takes place on the queen's side or the king's side, using the files given.
- The pieces involved are a king and a rook, neither of which may have moved since the beginning of the game.
- Neither the current position nor that formed by putting the king on the intermediate square over which it passes may be in check
- The king and the rook swap over, to the stated positions.

We add castling in using schema disjunction again:

$$Mover == PlainMover \lor Castler$$

43.8 Legal moves

A move is legal if it is carried out in accordance with *Mover* and does not leave the last player in check.

Legal

$$Mover$$

$$(player, pos') \notin check$$

$$legalMove == \{\ Legal \bullet g \mapsto g'\ \}$$

Legal play is the transitive closure of legal moves, that is, what we get by making any number of legal moves in succession.

$$legalPlay == legalMove^+$$

and we can characterise starting correctly:

$$legalGame == \text{successors}\, legalPlay\ initGame$$

The set of legal games is the image of the initial game through all possible legal plays.

43.9 Winning

We define the games that are in check, and hence checkmate.

$$checkGame == \{\ g : game \mid \text{last}\, g \in check\ \}$$
$$mate == checkGame \setminus \text{dom}\, legalMove$$

If mate is achieved, the game is finished, with the player in check having lost.

$$win : team \to \mathbb{P}\, game$$

$$win = \lambda\, t : team \bullet \{\ g : mate \mid turn(\text{last}\, g) \neq t\ \}$$

43.10 Draws

If no move is possible but the player is not in check we have stalemate.

$$stalemate == game \setminus checkGame \setminus \text{dom}\, legalMove$$

Stalemate is counted as a draw.

Another form of draw occurs where the board is so denuded that no mate is possible by means of legal play.

$$noMate == \{\ g : game \mid \neg\ (\ \exists\, g' : mate \bullet g \mapsto g' \in legalPlay\ )\ \}$$

Stalemate is in fact a special case of this.

A *repeatDraw* occurs where the same position is repeated three times.

$$repeatDraw == \{\ g : game \mid \exists\, i, j, k : \mathrm{dom}\, g \mid i < j < k \bullet g\ i = g\ j = g\ k\ \}$$

To consider the next form of draw, we need to categorise reversible moves.

$$\begin{aligned} reversible == \{\ & Legal \mid \\ & (\ \exists\, t : square \rightarrowtail square \bullet pos' = t \fatsemi pos\ ) \\ & \land\ pos \rhd \{player\ pawn\} = pos' \rhd \{player\ pawn\} \bullet \\ & g \mapsto g'\ \} \end{aligned}$$

This says that there is an injection that relates the old and new positions by mapping squares to squares, and also that the positions range-restricted to pawns are equal. Thus, a move in which no man is taken, and where no pawn is moved, is counted as reversible. Taking the transitive closure

$$revPlay == reversible^{+}$$

allows us to define

$$slowDraw == \{\ g, h : game \mid (g, h) \in revPlay \land \#(h \setminus g) = 50 \bullet h\ \}$$

which says that we have a draw where there have been fifty successive reversible moves.

So finally we can say

$$draw == stalemate \cup noMate \cup repeatDraw \cup slowDraw$$

43.11 Other endings

One aspect of the game that we have not modelled is the human protocol, although that would also be possible. This means, however, that we have not formalised the fact that the player and the opponent are separate persons each in control of their separate moves. Consequently we have not formalised the ending of a game by resignation, nor that of a draw by mutual agreement.

Another form of draw quoted in the books is "perpetual check". This also cannot be incorporated in our model because of the element of negotiation involved. The draw arises not when perpetual check is possible, which would be easy to formalise, but only if one of the players chooses to assert his or her intention to cause it. If the assertion is accepted, we can count that a draw by agreement. If not, the player can easily turn it into a *repeatDraw* or a *slowDraw*.

43.12 Optimal play

We have now given a complete description of legal play and its consequences, which could be refined into a game referee, for example. The purpose of the game, however, is to win, or if one cannot win, to draw. We define the function *winnable*, which gives the games that may be won by a particular team in a particular number of moves, assuming correct play for that team.

$$
\begin{array}{|l}
winnable : team \to \mathbb{N} \to \mathbb{P}\, game \\
\hline
\forall\, t : team \bullet winnable\ t\ 0 = win\ t \\
\forall\, t : team;\ n : \mathbb{N}_1 \bullet \\
\quad winnable\ t\ n = \{\ g : game \setminus draw \mid \\
\qquad (turn(\text{last}\ g) = t \Rightarrow \text{successors}\, legalMove\ g \cap winnable\ t(n-1) \neq \varnothing) \\
\qquad \land\ (turn(\text{last}\ g) \neq t \Rightarrow \text{successors}\, legalMove\ g \subseteq winnable\ t(n-1))\ \}
\end{array}
$$

We also define *advantage*, which gives the games that may be won in any number of moves.

$$advantage == \lambda\, t : team \bullet \bigcup \{\ n : \mathbb{N} \bullet winnable\ t\ n\ \}$$

evenGame is the set of games that are not at the mercy of either side.

$$evenGame == game \setminus advantage\ white \setminus advantage\ black$$

The three categories *evenGame*, *advantage white* and *advantage black* form a partition of *game*.

$$\vdash \langle \mathit{evenGame}, \mathit{advantage\ white}, \mathit{advantage\ black} \rangle \text{ partition } \mathit{game}$$

Optimal play can be defined as play that keeps the game in the same category as it was, since by the definitions a player can never move into a more favourable category; progress can only be made if one's opponent makes a mistake.

The definition of optimal play can feasibly be refined into a computer program only in endgame positions. In the early and middle games the size of the quantifications, although finite, is beyond the power of any computer. In fact it is an open question which category *initGame* belongs to. Although one might guess that it is a member of *evenGame* or *advantage white*, this has not been proved.

So, except for the endgame, the formalisation given for optimal play is notable in its uselessness. In order to get any further, we should have to make a formal model not only of the actual state of play, but also of our knowledge of the state of play, and of the actual computational power at our disposal. This might lead on to the heuristic methods actually used in play, like the concept of a "strong" or "weak" position, in contrast to the stark categories of complete knowledge, where a position is either won, lost or drawn. These are the concepts of real chess programs. A formal model of partial chess understanding has never been attempted, as far as we know. It might be quite illuminating, even if unreliable.

Part VII

Appendix

Appendix A

Diagrammatic conventions

A.1 Introduction

This chapter describes the various diagram conventions we use throughout the document.

A.2 Venn diagrams

In a real specification most sets have internal structure, so it is rare to have to draw diagrams of simple sets. However, on occasion, such diagrams can be a useful aid for visualising the effect of an operation.

We emphasise that the cardinality of a set is in general infinite, or indeterminate, and that finite sets are a special case (§15). In particular, a given set that is not further constrained (for example, by being defined using a free type) is neither provably finite nor infinite. This should be borne in mind when interpreting the (necessarily finite) diagrams.

Venn diagrams were proposed by the English logician John Venn, 1834–1923. A *Venn diagram* (figure A.1) is a sound way of diagramming one or more sets of the same type, where any internal structure that the elements may have is ignored. A Venn diagram consists of a rectangular box representing the whole type, X, containing a number of ellipses, each of which represents membership of some subset of the type. The name of the set represented by the ellipse can be written anywhere inside, or close outside, the ellipse, positioned such that it cannot be confused with that of another ellipse.

A set element can be marked as a small spot or cross within the relevant area of the diagram. The appearance of such a marked element on the diagram shows that the existence of a corresponding element in the set is consistent with the diagram,

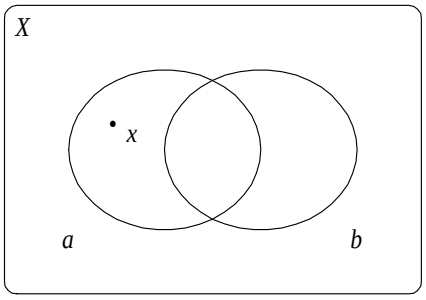


Figure A.1 An example of a Venn diagram, a diagrammatic convention for simple sets. The ellipses marked a and b represent subsets of the type X. x is an element in a but not in b; $x \in a$, $x \notin b$.

but it does not prove that such an element exists in all cases; a particular region of the diagram may be realised by the empty set.

In a properly formed diagram, each ellipse is drawn to overlap all others except where there is a demonstrable constraint to the contrary. The size, position, and orientation of each ellipse is unimportant; the only relevant factor is the pattern of overlap. The result of set operations may consist of separate areas (for example, see figure 13.13). That these resultant spaces may be neither elliptical, nor connected, is unimportant. If we consider specific points in a Venn diagram, the only relevant factor is which ellipses contain, or do not contain, them.

A Venn diagram is not just an example; it is a general diagram valid for all its possible instances. A properly formed Venn diagram is (very nearly) sufficient on its own as a proof, or as a counter-example in propositional calculus or set theory (for example, see figure 13.8).

A.3 Diagrams for relations

There are two diagrammatic conventions useful for sketching relations.

A.3.1 Venn diagrams

Relations can be sketched using two Venn diagrams, one for the source set, and one for the target set, with the relation shown as lines linking source elements to target elements (figure A.2).

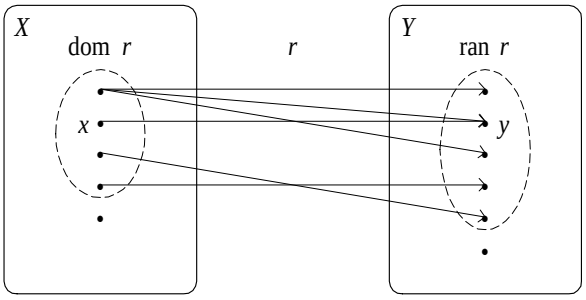


Figure A.2 Venn diagram convention for relations. The pair (x, y) is in the relation $r : X \leftrightarrow Y$.

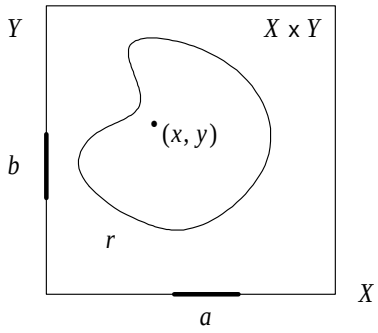


Figure A.3 Cartesian diagram convention for relations. The pair (x, y) is in the relation $r : X \leftrightarrow Y$. The set a is a named subset of X, and b is a named subset of Y.

So operations like domain and range restriction and subtraction can be represented by the appropriate set intersection or difference in one of the two subdiagrams.

A.3.2 Cartesian diagrams

The French philosopher René Descartes, 1596–1650, showed the utility of using two orthogonal coordinates in geometrical description. A *Cartesian diagram* (figure A.3) represents the Cartesian product set $X \times Y$ as a rectangular box whose horizontal base line represents the source set X, and whose vertical side line represents the target set Y. A relation r is a subset of $X \times Y$, and so is represented as an ellipse, or other shaped blob, within this rectangular $X \times Y$ plane. As with a Venn diagram, the shape of this blob has no significance. In our example diagrams we sometimes use a particular shaped blob to clarify the effect of an operation, but no significance should be read into such a shape.

A Cartesian diagram represents a simple set as a line segment, and subsets as subsegments of this line, drawn thicker and labelled appropriately. As with a Venn diagram's ellipse, the size and position of this line has no significance except to show the required pattern of overlap. A diagram may need to use a discontinuous line to represent the overlap; this discontinuity has no significance. The order of the elements along the line is not significant, and the finite extent of such a line should not be taken to imply that the set being represented is itself finite.

Dashed lines may be used to indicate the projection of some subset of X or Y on the $X \times Y$ plane. Individual pairs (x, y) may be marked; the relation may contain other pairs not so marked.

A Cartesian diagram, like a Venn diagram, is not just an example; it is a general diagram valid for all its possible instances.

A.4 Functions

The conventions that apply to relations are inherited by functions.

With Venn diagrams, the diagrams are of a similar appearance but there is only one range element corresponding to each domain element.

With Cartesian diagrams, the blob reduces to a line in the $X \times Y$ plane. Again, no particular order of the elements in the X and Y sets, or any properties of continuity, should be assumed to hold in the diagrams.

A.5 Homogeneous relations

A.5.1 Venn diagrams

Since the source and target sets are the same type, we can specialise the Venn diagram convention for general relations (figure A.4a).

We show the relation with arcs that emanate from 'source' elements in the ellipse representing the domain, curve back to the same rectangle, and end on 'sink' elements in the ellipse representing the range. Elements in the overlap region representing both the domain and range are both 'sources' and 'sinks'.

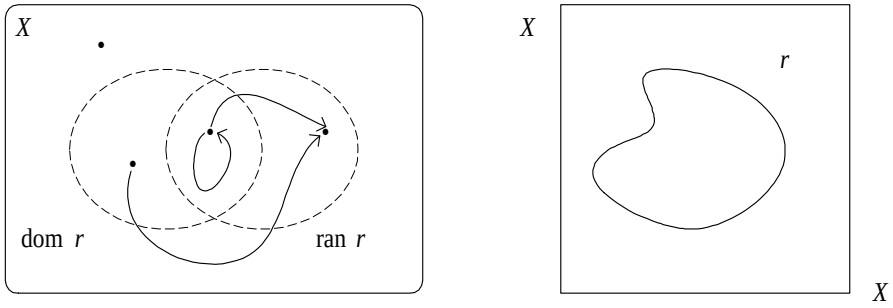


Figure A.4 Diagrammatic conventions for homogeneous relations: (a) Venn diagram (b) Cartesian diagram

A.5.2 Cartesian diagrams

We use the same convention for drawing the relation as part of the plane; we now have the $X \times X$ plane (figure A.4b).

Although we still assume no particular order of the elements along the X lines, we do now assume that it is the *same* order along each X line. Hence the diagonal represents all elements like (x, x).

A.6 Orders

A.6.1 Cartesian diagrams

The Cartesian diagram conventions for order relations are the same as for homogeneous binary relations, except that now the order of elements along the set lines is significant: we use the order defined by the relation.

For a partial order, not all elements need be comparable. However, if two elements are related by the order, $a \lessdot b$, we place a to the left of b on the horizontal axis, and a below b on the vertical axis. Hence any points representing a pair in the order fall in the upper-left triangle only; for a total order, where every pair is comparable, the points fill the upper-left triangle (and hence the diagram shows nothing interesting).

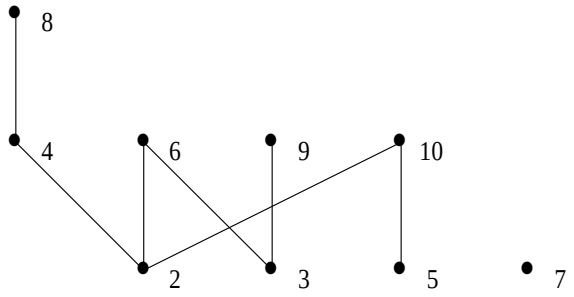


Figure A.5 Network diagram convention for orders: the numbers 2..10 partially ordered by 'is a factor of'.

A.6.2 Network diagrams

More useful for drawing orders than Venn diagrams or Cartesian diagrams is the *network diagram* convention (figure A.5). This is a further convention for Venn diagrams that omits much 'clutter' that is common to orders.

A network diagram includes only the *intransitive residue*. The elements in the order are drawn on the page so that b is higher on the page than a if $a \lessdot b$. b is joined to a by a line if there is no c such that $a \lessdot c \lessdot b$. Hence we can omit the arrowheads from the arcs, because they always point up the page, we can omit the transitive arcs, and we can omit the domain and range ellipses, because all the leaves are at the bottom and all the roots at the top.

Appendix B

Summary of operator templates

In this appendix we summarise the various operator templates defined in earlier chapters.

relation ($\_ \notin \_$)
relation ($\_ \neq \_$)
function ($\_^{2\times}$)

function 30 leftassoc ($\_ \cup \_$)
function 40 leftassoc ($\_ \cap \_$)
function 30 leftassoc ($\_ \setminus \_$)
function 30 leftassoc ($\_ \ominus \_$)

relation ($\_ \subseteq \_$)
relation ($\_ \subset \_$)

generic ($\mathbb{F}\ \_$)
relation (finite $\_$)

generic 5 rightassoc ($\_ \leftrightarrow \_$)
generic 5 rightassoc ($\_ \nrightarrow \_$)
function 10 leftassoc ($\_ \mapsto \_$)
function ($\_^{\sim}$)

function 65 rightassoc ($\_ \lhd \_$)
function 65 rightassoc ($\_ \ndomrestrict \_$)
function 60 leftassoc ($\_ \rhd \_$)
function 60 leftassoc ($\_ \nrangerestrict \_$)

function $(\_(\!|\ \_\ |\!)\_)$

relation $(\_ \approx \_)$
function 50 leftassoc $(\_ \oplus \_)$
function 40 leftassoc $(\_ \fatsemi \_)$
function 40 leftassoc $(\_ \circ \_)$
function 40 leftassoc $(\_ \overrightarrow{\fatsemi} \_)$
function 40 leftassoc $(\_ \overleftarrow{\fatsemi} \_)$

relation $(\_$ functionalAt $\_)$
generic 5 rightassoc $(\_ \nrightarrow \_)$
generic 5 rightassoc $(\_ \rightarrow \_)$
generic 5 rightassoc $(\_ \twoheadrightarrow \_)$
generic 5 rightassoc $(\_ \twoheadrightarrow\!\!\!\!\!\!\rightarrow \_)$
generic 5 rightassoc $(\_ \twoheadrightarrow \_)$
generic 5 rightassoc $(\_ \twoheadrightarrow\!\!\!\!\!\!\rightarrow \_)$
generic 5 rightassoc $(\_ \rightarrowtail \_)$
generic 5 rightassoc $(\_ \rightarrowtail \_)$
generic 5 rightassoc $(\_ \twoheadrightarrow\!\!\!\!\!\!\rightarrow \_)$
generic 5 rightassoc $(\_ \twoheadrightarrow\!\!\!\!\!\!\rightarrow \_)$
generic 5 rightassoc $(\_ \twoheadrightarrow \_)$
generic 5 rightassoc $(\_ \twoheadrightarrow\!\!\!\!\!\!\rightarrow \_)$
function 30 leftassoc $(\_ \diamond_x \_)$
function 30 leftassoc $(\_ \diamond_y \_)$

function 30 leftassoc $(\_ \diamond \_)$
generic (idempotent $\_$)

generic (wellRooted _)
generic (id _)
generic (reflexive _)
generic (irreflexive _)
generic (symmetric _)
generic (antisymmetric _)
generic (transitive _)
function ($\_^{+}$)
function ($\_^{*}$)
generic (intransitive _)
generic 80(locallyFiniteOut _)
generic 80(locallyFiniteIn _)
generic 80(locallyFinite _)
generic (equivalence _)
generic (acyclic _)

generic (stronglyConnected _)
generic (connected _)
generic (forest _)
generic (tree _)

generic (order _)
generic (reflexiveOrder _)
generic (irreflexiveOrder _)
generic (totalOrder _)
generic (reflexiveTotalOrder _)
generic (irreflexiveTotalOrder _)
generic (preorder _)
generic (wellOrder _)

function 30 leftassoc ($\_ \dot{+} \_$)
function 40 leftassoc ($\_ \dot{*} \_$)
function ($\dot{-} \_$)
function ($\_^{\dot{-}1}$)

relation (_ < _)
relation (_ ≤ _)
function 40 leftassoc (_ * _)
function (-_)
function (_⁻_)
function (_⁺)
function (_⊕)
function (_±)
function (_⁻¹)
function 30 leftassoc (_ − _)
function 40 leftassoc (_ ÷ _)
relation (_ ≥ _)
relation (_ > _)

function (sign _)
function (abs _)
function (⌊ _ ⌋)
function (⌈ _ ⌉)
function 40 leftassoc (_ div _)
function 40 leftassoc (_ mod _)
function 20 leftassoc (_ . . _)
function (# _)
generic ($\vec{\#}$ _)
function (min _)
function (max _)
relation (coprime _)
function ($\sqrt{\_}$)

function (_⁻)

relation (disjoint _)
relation (_ partition _)

function (factorial _)
function 45 rightassoc (_ ** _)
function (exp _)
function (ln _)
function (log _)
function (sin _)
function (cos _)

generic (stream _)
generic (istream _)
generic (sequence _)
generic (isequence _)
generic (str _)
generic (istr _)
generic (seq _)
generic (iseq _)
function 60 leftassoc (_ shift _)
function (_⟨, , ⟩)
function (⟨, , ⟩)

function 30 leftassoc (_ ⌢ _)
generic (enumerableOrder _)
generic (reflexiveEnumerableOrder _)
generic (irreflexiveEnumerableOrder _)

relation (_ prefix _)
relation (_ suffix _)
relation (_ infix _)

function 40 leftassoc (_ ↑ _)
function 40 leftassoc (_ ↓ _)
function 40 leftassoc (_ ↾ _)
function 40 leftassoc (_ ⇂ _)

generic (bag _)
function 30 leftassoc (_ ⊎ _)

Appendix C

Bibliography

[Abrial 1980]
Jean-Raymond Abrial. *The Specification Language Z: Syntax and Semantics*. Programming Research Group, Oxford University Computing Laboratory, 1980.

[Arthan 1991]
R. D. Arthan. On formal specification of a proof tool. In S. Prehn and W. J. Toetenel, editors, *VDM'91: Formal Software Development Methods, Volume 1: Conference Contributions*, volume 551 of *LNCS*, pages 356–370. Springer, 1991.

[Barden *et al.* 1994]
Rosalind Barden, Susan Stepney, and David Cooper. *Z in Practice*. Prentice Hall, 1994.

[Bhattacharya *et al.* 1994]
P. B. Bhattacharya, S. K. Jain, and S. R. Nagpaul. *Basic Abstract Algebra*. Cambridge University Press, 2nd edition, 1994.

[Bird & Wadler 1988]
Richard Bird and Philip Wadler. *Introduction to Functional Programming*. Prentice Hall, 1988.

[Bishop 1995]
Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[Bollobás 1986]
Béla Bollobás, editor. *Littlewood's Miscellany*. Cambridge University Press, 1986.

[Bowen & Hinchey 1995]
Jonathan P. Bowen and Michael G. Hinchey, editors. *ZUM'95: 9th International Conference of Z Users, Limerick*, volume 967 of *LNCS*. Springer, 1995.

[Bradley 1991]
Keith R. Bradley. *Discovering the Roman Family*. Oxford University Press, 1991.

[Davey & Priestly 1990]
B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[Dijkstra 1976]
Edsgar W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[Ebbinghaus *et al.* 1991]
H.-D. Ebbinghaus, H. Hermes, F. Hirzebruch, M. Koecher, K. Mainzer, J. Neukirch, A. Prestel, and R. Remmert. *Numbers*. Springer, 1991.

[Enderton 1977]
H. B. Enderton. *Elements of Set Theory*. Academic Press, 1977.

[Fox 1967]
Robin Fox. *Kinship and Marriage*. Pelican, 1967.

[Fraleigh 1994]
John B. Fraleigh. *A First Course in Abstract Algebra*. Addison Wesley, 5 edition, 1994.

[Halmos 1960]
Paul R. Halmos. *Naive Set Theory*. Van Nostrand Reinhold, 1960.

[Harwood 1995]
Will Harwood. A Zola tutorial. Technical report, Imperial Software Technology, Cambridge, UK, 1995.

[Hayes 1987]
Ian J. Hayes, editor. *Specification Case Studies*. Prentice Hall, 1987.

[Hayes 1990]
Ian J. Hayes. Multi-sets and multi-relations in Z with an application to a bill-of-materials system. Technical Report UQCS-176, Department of

Computer Science, University of Queensland, July 1990.

[Hayes 1993]
Ian J. Hayes, editor. *Specification Case Studies*. Prentice Hall, 2nd edition, 1993.

[Heinlein 1959]
Robert A. Heinlein. *The Unpleasant Profession of Jonathan Hoag*. Gnome, 1959.

[Hervey 1979]
Sandor G. J. Hervey. *Axiomatic Semantics: a theory of linguistic semantics*. Scottish Academic Press, 1979.

[ISO-Z 2002]
ISO/IEC 13568. *Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics: International Standard*, 2002.
http://www.iso.org/iso/en/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip.

[Iverson 1962]
Kenneth E. Iverson. *A Programming Language*. Wiley, 1962.

[Jones 1992]
Roger B. Jones. *Proposal to allow general use of expressions as schemas in Z*. ICL, 1992.

[Jordan *et al.* 1991]
David T. Jordan, John A. McDermid, and Ian Toyn. CADiZ – computer aided design in Z. In John E. Nicholls, editor, *Proceedings of the 5th Annual Z User Meeting, Oxford 1990*, Workshops in Computing, pages 93–104. Springer, 1991.

[Kaye 1986]
Kenneth Kaye. Non relatives. *Journal of Irreproducible Results*, 31(3):13–14, Feb/Mar 1986.

[Kazic *et al.* 1985]
B. Kazic, R. D. Keene, and K. A. Lim, editors. *The Official Laws of Chess*. Batsford, 1985.

[King *et al.* 1988]
Steve King, Ib Holm Sørenson, and James C. P. Woodcock. Z: Grammar and concrete and abstract syntaxes. Technical Monograph PRG-68,

Programming Research Group, Oxford University Computing Laboratory, 1988.

[Machado & Meira 1995]
Patrícia Duarte de Lima Machado and Silvio Lemos Meira. On the use of formal specifications in the design and simulation of artificial neural networks. In [Bowen & Hinchey 1995], pages 63–82.

[Mommsen *et al.* 1985]
Theodor Mommsen, Paul Krueger, and Alan Watson, editors. *The Digest of Justinian*, volume III. University of Pennsylvania Press, 1985.

[Potter *et al.* 1991]
Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specification and Z*. Prentice Hall, 1991.

[Reingold & Dershowitz 2001]
Edward M. Reingold and Nachum Dershowitz. *Calendrical Calculations: the millennium edition*. CUP, 2001.

[Russell 1929]
Bertrand Russell. *Mysticism and Logic*. 1929.

[Spivey 1988]
J. Michael Spivey. *Understanding Z: a specification language and its formal semantics*. Cambridge University Press, 1988.

[Spivey 1989]
J. Michael Spivey. *The Z Notation: a Reference Manual*. Prentice Hall, 1989.

[Spivey 1992]
J. Michael Spivey. *The Z Notation: a Reference Manual*. Prentice Hall, 2nd edition, 1992.

[Steele, Jr. 1990]
Guy L. Steele, Jr. *Common Lisp: the Language*. Digital Press, second edition, 1990.

[Stepney *et al.* 2003]
Susan Stepney, Fiona Polack, and Ian Toyn. A Z patterns catalogue I: specification and refactorings, v0.1. Technical Report YCS-2003-349, Department of Computer Science, University of York, January 2003.

[Stewart 1973]
Ian Stewart. *Galois Theory*. Chapman and Hall, 1973.

[Sufrin *et al.* 1984]
Bernard A. Sufrin, C. Carroll Morgan, Ib Holm Sørensen, and Ian J. Hayes. *Notes for a Z handbook*. Programming Research Group, Oxford University Computing Laboratory, 1984.

[Sufrin 1986]
Bernard A. Sufrin. *Z Handbook - draft 1.1*. Programming Research Group, Oxford University Computing Laboratory, March 1986.

[Suppes 1972]
Patrick Suppes. *Axiomatic Set Theory*. Dover, 1972.

[Toyn 1996]
Ian Toyn. Formal reasoning in the Z notation using CADiZ. In N. A. Merriam, editor, *2nd International Workshop on User Interface Design for Theorem Proving Systems*, 1996.

[Toyn 1998]
Ian Toyn. Innovations in the notation of Standard Z. In Jonathan P. Bowen, Andreas Fett, and Michael G. Hinchey, editors, *ZUM'98: 11th International Conference of Z Users, Berlin*, volume 1493 of *LNCS*, pages 193–213. Springer, 1998.

[Unicode 1996]
ISO/IEC 10646. *Unicode: worldwide character encoding: Committee Draft*, 1996.

[Valentine *et al.* 2000]
Samuel H. Valentine, Ian Toyn, Susan Stepney, and Steve King. Type-constrained generics for z. In Jonathan P. Bowen, Steve Dunne, Andy Galloway, and Steve King, editors, *ZB2000: 1st International Conference of B and Z Users, York*, volume 1878 of *LNCS*, pages 250–263. Springer, 2000.

[Valentine 1993a]
Samuel H. Valentine. Enhancing the Z mathematical toolkit. Technical Report UBC 93/3, University of Brighton, Department of Computing, February 1993.

[Valentine 1993b]

Samuel H. Valentine. Putting numbers into the mathematical toolkit. In Jonathan P. Bowen and John E. Nicholls, editors, *Proceedings of the 7th Annual Z User Meeting, London*, Workshops in Computing. Springer, 1993.

[Valentine 1995]
Samuel H. Valentine. Equal rights for schemas in Z. In [Bowen & Hinchey 1995], pages 203–223.

[Woodcock & Davies 1996]
Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement, and Proof.* Prentice Hall, 1996.

[Wordsworth 1992]
John B. Wordsworth. *Software Development with Z.* Addison-Wesley, 1992.

Appendix D

Proofs

Proof of law 6.14, that universal quantification over the empty set is always true

$$
\begin{array}{lr}
(\ \forall x : \varnothing[X] \bullet P\ ) & \\
\quad \Leftrightarrow (\ \forall x : X \mid x \in \varnothing[X] \bullet P\ ) & \text{normalise} \\
\quad \Leftrightarrow (\ \forall x : X \mid \mathit{false} \bullet P\ ) & \text{prop. } \varnothing \\
\quad \Leftrightarrow (\ \forall x : X \bullet \mathit{false} \Rightarrow P\ ) & \text{law 6.12} \\
\quad \Leftrightarrow (\ \forall x : X \bullet \mathit{true}\ ) & \text{prop. } \Rightarrow \\
\quad \Leftrightarrow \mathit{true} & \text{law 6.11}
\end{array}
$$

■

Proof of law 9.10, that power set distributes through intersection:

$$
\begin{array}{lr}
\mathbb{P}(\bigcap \alpha) & \\
\quad = \{\ b : \mathbb{P}\, X \mid b \subseteq \bigcap \alpha)\ \} & \text{law 9.7} \\
\quad = \{\ b : \mathbb{P}\, X \mid (\ \forall x : b \bullet x \in \bigcap \alpha)\ )\ \} & \text{law 14.1} \\
\quad = \{\ b : \mathbb{P}\, X \mid (\ \forall x : b \bullet \forall a : \alpha \bullet x \in a\ )\ \} & \text{defn. } \bigcap \\
\quad = \{\ b : \mathbb{P}\, X \mid (\ \forall a : \alpha \bullet \forall x : b \bullet x \in a\ )\ \} & \text{rearrange} \\
\quad = \{\ b : \mathbb{P}\, X \mid (\ \forall a : \alpha \bullet b \subseteq a\ )\ \} & \text{defn. } \subseteq \\
\quad = \{\ b : \mathbb{P}\, X \mid (\ \forall a : \alpha \bullet b \in \mathbb{P}\, a\ )\ \} & \text{property } \mathbb{P} \\
\quad = \{\ b : \mathbb{P}\, X \mid (\ \forall \beta : \{\ a : \alpha \bullet \mathbb{P}\, a\ \} \bullet b \in \beta\ )\ \} & \text{change variable} \\
\quad = \bigcap \{\ a : \alpha \bullet \mathbb{P}\, a\ \} & \text{defn. } \bigcap
\end{array}
$$

■

Proof of law 9.13, that power set never distributes through (symmetric) set difference:

Every power set contains at least the empty set, therefore the (symmetric) difference of two power sets does not, therefore the (symmetric) difference cannot equal the power set of anything.

Proof of law 12.1, that $\top$ is *true*

$$
\begin{array}{rl}
\top & \\
\quad \Rightarrow [\ \mid \mathit{true}] & \text{definition} \\
\quad \Rightarrow \theta[\ \mid \mathit{true}\ ] \in [\ \mid \mathit{true}\ ] & \text{expand schema predicate} \\
\quad \Rightarrow \theta[\ \mid \mathit{true}\ ] \in \{(\!|\ \ |\!)\} & \text{expand empty schema} \\
\quad \Rightarrow (\!|\ \ |\!) \in \{(\!|\ \ |\!)\} & \text{expand } \theta \\
\quad \Rightarrow \mathit{true} & \text{expand membership}
\end{array}
$$

■

and that $\bot$ is *false*

$$
\begin{array}{rl}
\bot & \\
\quad \Rightarrow [\ \mid \mathit{false}\ ] & \text{definition} \\
\quad \Rightarrow \theta[\ \mid \mathit{false}\ ] \in [\ \mid \mathit{false}\ ] & \text{expand schema predicate} \\
\quad \Rightarrow \theta[\ \mid \mathit{false}\ ] \in \{\ \} & \text{expand empty schema} \\
\quad \Rightarrow \mathit{false} & \text{expand membership}
\end{array}
$$

■

Proof of law 13.6, that generalised union of a set comprehension can be reduced to a set comprehension

$$
\begin{array}{rl}
\bigcup\{\ a : \alpha \bullet f\ a\ \} & \\
\quad = \{\ y : Y \mid (\ \exists b : \{\ a : \alpha \bullet f\ a\ \} \bullet y \in b\ )\ \} & \text{defn. } \bigcup \\
\quad = \{\ y : Y \mid (\ \exists b : \mathbb{P}\, Y \bullet b \in \{\ a : \alpha \bullet f\ a\ \} \land y \in b\ )\ \} & \text{expand} \\
\quad = \{\ y : Y \mid (\ \exists b : \mathbb{P}\, Y \bullet (\ \exists a : \alpha \bullet b = f\ a\ ) \land y \in b\ )\ \} & \text{rearrange} \\
\quad = \{\ y : Y \mid (\ \exists a : \alpha \bullet y \in f\ a\ )\ \} & \text{rearrange; eliminate } b
\end{array}
$$

■

Proof of law 13.7, that nested generalised unions can be flattened.

$$
\begin{array}{rl}
\bigcup\{\ a : \alpha \bullet \bigcup\{\ b : \beta \bullet f(a,b)\ \}\ \} & \\
\quad = \bigcup\{\ a : \alpha \bullet \{\ z' : Z \mid (\ \exists b : \beta \bullet z' \in f(a,b)\ )\ \}\ \} & \text{simplify} \\
\quad = \{\ z : Z \mid (\ \exists a : \alpha \bullet z \in \{\ z' : Z \mid (\ \exists b : \beta \bullet z' \in f(a,b)\ )\ \}\ )\ \} & \text{simplify} \\
\quad = \{\ z : Z \mid (\ \exists a : \alpha \bullet (\ \exists b : \beta \bullet z \in f(a,b)\ )\ )\ \} & \text{simplify} \\
\quad = \bigcup\{\ a : \alpha;\ b : \beta \bullet f(a,b)\ \} & \bigcup\text{-reduction law 13.6}
\end{array}
$$

■

Proof of law 13.9, that $\cap$ distributes through $\bigcup$

$$
\begin{array}{rl}
\bigcup \alpha \cap \bigcup \beta & \\
\quad = \{\ x : X \mid (\ \exists a : \alpha \bullet x \in a\ ) \land (\ \exists b : \beta \bullet x \in b\ )\ \} & \text{defn. } \bigcup, \cap \\
\quad = \{\ x : X \mid (\ \exists a : \alpha;\ b : \beta \bullet x \in a \cap b\ )\ \} & \exists \text{ law; defn } \cap \\
\quad = \bigcup\{\ a : \alpha;\ b : \beta \bullet a \cap b\ \} & \bigcup\text{-reduction law 13.6}
\end{array}
$$

■

Proof of law 13.11, that $\bigcup(\bigcup(\!|\ A\ |\!)) = \bigcup(\bigcup A)$

$$
\begin{array}{ll}
\bigcup(\bigcup A) & \\
\quad = \bigcup\{\ a : \mathbb{P}\, X \mid (\ \exists \alpha : A \bullet a \in \alpha\ )\ \} & \text{defn. } \bigcup \\
\quad = \{\ x : X \mid (\ \exists a : \mathbb{P}\, X \mid (\ \exists \alpha : A \bullet a \in \alpha\ ) \bullet x \in a\ )\ \} & \bigcup \text{ law 13.6} \\
\quad = \{\ x : X \mid (\ \exists a : \mathbb{P}\, X \bullet \exists \alpha : A \bullet a \in \alpha \wedge x \in a\ )\ \} & \exists \text{ law} \\
\quad = \{\ x : X \mid (\ \exists \alpha : A \bullet \exists a : \alpha \bullet x \in a\ )\ \} & \text{rearrange} \\
\quad = \{\ x : X \mid (\ \exists \alpha : A \bullet x \in \{\ y : Y \mid (\ \exists a : \alpha \bullet y \in a\ )\ \}\ )\ \} & \text{law set comp.} \\
\quad = \{\ x : X \mid (\ \exists \alpha : A \bullet x \in \bigcup a\ )\ \} & \text{defn. } \bigcup \\
\quad = \bigcup\{\ \alpha : A \bullet \bigcup a\ \} & \bigcup \text{ law 13.6} \\
\quad = \bigcup(\bigcup(\!|\ A\ |\!)) & \_(\!|\ \_\ |\!) \text{ law 19.3}
\end{array}
$$

■

Proof of law 13.14, that distribution occurs when pairwise disjoint

$$
\begin{array}{ll}
\bigcup\alpha \cap \bigcup\beta & \\
\quad = \{\ x : X \mid (\ \exists a : \alpha \bullet x \in a\ ) \wedge (\ \exists a : \beta \bullet x \in a\ )\ \} & \text{defn. } \bigcup, \cap \\
\quad = \{\ x : X \mid (\ \exists a : \mathbb{P}\, X \mid a \in \alpha \wedge a \in \beta \bullet x \in a\ )\ \} & \text{hyp. } a \neq b \Rightarrow a \cap b = \varnothing \\
\quad = \bigcup\{\ a : \mathbb{P}\, X \mid a \in \alpha \wedge a \in \beta\ \} & \bigcup \text{ law 13.6} \\
\quad = \bigcup(\alpha \cap \beta) & \text{defn. } \cap
\end{array}
$$

■

Proof of law 13.22, that generalised intersection of a set comprehension can be reduced to a set comprehension

$$
\begin{array}{ll}
\bigcap\{\ a : \alpha \bullet f\ a\ \} & \\
\quad = \{\ y : Y \mid (\ \forall b : \{\ a : \alpha \bullet f\ a\ \} \bullet y \in b\ )\ \} & \text{defn. } \bigcap \\
\quad = \{\ y : Y \mid (\ \forall b : \mathbb{P}\, Y \mid b \in \{\ a : \alpha \bullet f\ a\ \} \bullet y \in b\ )\ \} & \text{expand} \\
\quad = \{\ y : Y \mid (\ \forall b : \mathbb{P}\, Y \mid (\ \exists a : \alpha \bullet b = f\ a\ ) \bullet y \in b\ )\ \} & \text{rearrange} \\
\quad = \{\ y : Y \mid (\ \forall b : \mathbb{P}\, Y \bullet (\ \forall a : \alpha \bullet b \neq f\ a\ ) \vee y \in b\ )\ \} & \text{expand } \Rightarrow \\
\quad = \{\ y : Y \mid (\ \forall b : \mathbb{P}\, Y;\ a : \alpha \mid b = f\ a \bullet y \in b\ )\ )\ \} & \text{rearrange} \\
\quad = \{\ y : Y \mid (\ \forall a : \alpha \bullet y \in f\ a\ )\ \} & \text{eliminate } b
\end{array}
$$

■

Proof of law 13.23, that nested generalised intersections can be flattened

$$
\begin{array}{ll}
\bigcap\{\ a : \alpha \bullet \bigcap\{\ b : \beta \bullet f(a,b)\ \}\ \} & \\
\quad = \bigcap\{\ a : \alpha \bullet \{\ z' : Z \mid (\ \forall b : \beta \bullet z' \in f(a,b)\ )\ \}\ \} & \text{simplify} \\
\quad = \{\ z : Z \mid (\ \forall a : \alpha \bullet z \in \{\ z' : Z \mid (\ \forall b : \beta \bullet z' \in f(a,b)\ )\ \}\ )\ \} & \text{simplify} \\
\quad = \{\ z : Z \mid (\ \forall a : \alpha \bullet (\ \forall b : \beta \bullet z \in f(a,b)\ )\ )\ \} & \text{simplify} \\
\quad = \bigcap\{\ a : \alpha;\ b : \beta \bullet f(a,b)\ \} & \bigcap\text{-reduction law 13.22}
\end{array}
$$

■

Proof of law 13.24, that $\cup$ distributes through $\bigcap$:

$$
\begin{array}{ll}
\bigcap \alpha \cup \bigcap \beta & \\
\quad = \{\ x : X \mid (\ \forall a : \alpha \bullet x \in a\ ) \vee (\ \forall b : \beta \bullet x \in b\ )\ \} & \text{defn. } \bigcap, \cup \\
\quad = \{\ x : X \mid (\ \forall a : \alpha;\ b : \beta \bullet x \in a \vee x \in b\ )\ \} & \forall \text{ law 6.16} \\
\quad = \{\ x : X \mid (\ \forall a : \alpha;\ b : \beta \bullet x \in a \cup b\ )\ \} & \text{defn. } \cup \\
\quad = \bigcap \{\ a : \alpha;\ b : \beta \bullet a \cup b\ \} & \bigcap\text{-reduction law 13.22}
\end{array}
$$

■

Proof of law 13.28, that $\bigcap(\bigcap(\!|\ A\ |\!)) = \bigcap(\bigcup A)$

$$
\begin{array}{ll}
\bigcap(\bigcup A) & \\
\quad = \bigcap \{\ a : \mathbb{P}\, X \mid (\ \exists \alpha : A \bullet a \in \alpha\ )\ \} & \text{defn. } \bigcup \\
\quad = \{\ x : X \mid (\ \forall a : \mathbb{P}\, X \mid (\ \exists \alpha : A \bullet a \in \alpha\ ) \bullet x \in a\ )\ \} & \text{law } \bigcap \\
\quad = \{\ x : X \mid (\ \forall a : \mathbb{P}\, X \bullet (\ \forall \alpha : A \bullet a \notin \alpha\ ) \vee x \in a\ )\ \} & \text{expand } \Rightarrow \\
\quad = \{\ x : X \mid (\ \forall a : \mathbb{P}\, X \bullet \forall \alpha : A \mid a \in \alpha \bullet x \in a\ )\ \} & \text{law } \forall \\
\quad = \{\ x : X \mid (\ \forall \alpha : A \bullet \forall a : \alpha \bullet x \in a\ )\ \} & \text{rearrange} \\
\quad = \{\ x : X \mid (\ \forall \alpha : A \bullet x \in \{\ y : Y \mid (\ \forall a : \alpha \bullet y \in a\ )\ \}\ )\ \} & \text{law set comp.} \\
\quad = \{\ x : X \mid (\ \forall \alpha : A \bullet x \in \bigcap a\ )\ \} & \text{defn. } \bigcap \\
\quad = \bigcap \{\ \alpha : A \bullet \bigcap a\ \} & \text{law } \bigcap \\
\quad = \bigcap(\bigcap(\!|\ A\ |\!)) & \text{law } \_(\!|\ \_\ |\!)
\end{array}
$$

■

Proof of law 13.44, that distribution through $\setminus$ gives distribution through $\cap$:

$$
\begin{array}{ll}
f(a \cap b) & \\
\quad = f(a \setminus (a \setminus b)) & \text{identity} \\
\quad = f\ a \setminus f(a \setminus b) & \text{hypothesis} \\
\quad = f\ a \setminus (f\ a \setminus f\ b) & \text{hypothesis} \\
\quad = f\ a \cap f\ b & \text{identity}
\end{array}
$$

■

Proof of law 16.6, that $(Y \times X) \setminus r^{\sim} = ((X \times Y) \setminus r)^{\sim}$

$$
\begin{array}{ll}
((X \times Y) \setminus r)^{\sim} & \\
\quad = \{\ x : X;\ y : Y \mid x \mapsto y \in (X \times Y) \setminus r \bullet y \mapsto x\ \} & \text{defn. } \_^{\sim} \\
\quad = \{\ x : X;\ y : Y \mid x \mapsto y \in (X \times Y) \wedge x \mapsto y \notin r \bullet y \mapsto x\ \} & \text{defn. } \setminus \\
\quad = \{\ y : Y;\ x : X \mid y \mapsto x \in (Y \times X) \wedge y \mapsto x \notin r^{\sim}\ \} & \text{rearrange} \\
\quad = (Y \times X) \setminus r^{\sim} & \text{defn. } \setminus
\end{array}
$$

■

Proof of law 16.7, that $\_^\sim$ distributes through $\bigcup$

$$
\begin{array}{ll}
(\bigcup \rho)^\sim & \\
\quad = \{\ x : X;\ y : Y \mid (\ \exists\, r : \rho \bullet x \mapsto y \in r\ )\ \}^\sim & \text{defn. } \bigcup \\
\quad = \{\ y : Y;\ x : X \mid (\ \exists\, r : \rho \bullet x \mapsto y \in r\ )\ \} & \text{defn. } {}^\sim \\
\quad = \{\ y : Y;\ x : X \mid (\ \exists\, s : \rho^\sim \bullet y \mapsto x \in s\ )\ \} & \text{prop. } {}^\sim \\
\quad = \bigcup(\rho^\sim) &
\end{array}
$$

■

Proof of law 16.8, that $\_^\sim$ distributes through $\setminus$

$$
\begin{array}{ll}
(r \setminus s)^\sim & \\
\quad = \{\ x : X;\ y : Y \mid x \mapsto y \in r \land x \mapsto y \notin s\ \}^\sim & \text{defn. } \setminus \\
\quad = \{\ y : Y;\ x : X \mid x \mapsto y \in r \land x \mapsto y \notin s\ \} & \text{defn. } {}^\sim \\
\quad = \{\ y : Y;\ x : X \mid y \mapsto x \in r^\sim \land y \mapsto x \notin s^\sim\ \} & \text{defn. } {}^\sim \\
\quad = r^\sim \setminus s^\sim &
\end{array}
$$

■

Proof of law 17.3, that *dom* distributes through $\bigcup$

$$
\begin{array}{ll}
\bigcup(\mathrm{dom}(\!|\ \rho\ |\!)) & \\
\quad = \bigcup\{\ r : \rho \bullet \mathrm{dom}\, r\ \} & \text{expand } \_(\!|\ \_\ |\!) \\
\quad = \{\ x : X \mid (\ \exists\, a : \{\ r : \rho \bullet \mathrm{dom}\, r\ \} \bullet x \in a\ )\ \} & \text{defn. } \bigcup \\
\quad = \{\ x : X;\ a : \mathbb{P}\, X \mid a \in \{\ r : \rho \bullet \mathrm{dom}\, r\ \} \land x \in a \bullet x\ \} & \text{rearrange} \\
\quad = \{\ x : X;\ a : \mathbb{P}\, X \mid (\ \exists\, r : \rho \bullet a = \mathrm{dom}\, r\ ) \land x \in a \bullet x\ \} & \text{rearrange} \\
\quad = \{\ x : X;\ a : \mathbb{P}\, X;\ r : \rho \mid a = \mathrm{dom}\, r \land x \in a \bullet x\ \} & \text{rearrange} \\
\quad = \{\ x : X;\ r : \rho \mid x \in \mathrm{dom}\, r \bullet x\ \} & \text{eliminate } a \\
\quad = \{\ x : X;\ y : Y;\ r : \rho \mid x \mapsto y \in r \bullet x\ \} & \text{defn. dom} \\
\quad = \{\ x : X;\ y : Y \mid (\ \exists\, r : \rho \bullet x \mapsto y \in r\ ) \bullet x\ \} & \text{rearrange} \\
\quad = \mathrm{dom}(\bigcup \rho) & \text{defn. dom, } \bigcup
\end{array}
$$

■

Proof of law 17.6, that of distribution property of $\setminus$ through *dom*.

$$
\begin{array}{ll}
r = (r \setminus s) \cup (r \cap s) & \text{closure} \\
\mathrm{dom}\, r = \mathrm{dom}((r \setminus s) \cup (r \cap s)) & \\
\quad = \mathrm{dom}(r \setminus s) \cup \mathrm{dom}(r \cap s) & \text{dist.} \\
\mathrm{dom}\, r \setminus \mathrm{dom}\, s = (\mathrm{dom}(r \setminus s) \cup \mathrm{dom}(r \cap s)) \setminus \mathrm{dom}\, s & \\
\quad = \mathrm{dom}(r \setminus s) \setminus \mathrm{dom}\, s \cup \mathrm{dom}(r \cap s) \setminus \mathrm{dom}\, s & \text{dist.} \\
\quad = \mathrm{dom}(r \setminus s) \setminus \mathrm{dom}\, s & \text{subset-order pres.}
\end{array}
$$

■

Proof of law 18.5, that $\_ \lhd \_$ distributes through $\bigcup$ on both arguments

$$
\begin{array}{ll}
\bigcup \alpha \lhd \bigcup \rho & \\
\quad = (\bigcup \alpha \times Y) \cap \bigcup \rho & \text{rewrite } \lhd \text{ law 18.4} \\
\quad = \bigcup \{\ r : \rho \bullet (\bigcup \alpha \times Y) \cap r\ \} & \text{dist. } \cap - \bigcup \text{ law} \\
\quad = \bigcup \{\ r : \rho \bullet \bigcup \{ a : \alpha \bullet a \times Y\ \} \cap r\ \} & \text{dist. } \bigcup - \times \text{ law} \\
\quad = \bigcup \{\ r : \rho \bullet \bigcup \{ a : \alpha \bullet (a \times Y) \cap r\ \}\ \} & \text{dist. } \bigcup - \cap \text{ law} \\
\quad = \bigcup \{\ r : \rho \bullet \bigcup \{ a : \alpha \bullet a \lhd r\ \}\ \} & \text{rewrite } \lhd \text{ law 18.4} \\
\quad = \bigcup \{\ a : \alpha;\ r : \rho \bullet a \lhd r\ \} & \text{flatten } \bigcup - \bigcup \text{ law}
\end{array}
$$

■

Proof of law 18.6, that $\_ \lhd \_$ distributes through $\bigcap$ on both arguments

$$
\begin{array}{ll}
\bigcap \alpha \lhd \bigcap \rho & \\
\quad = (\bigcap \alpha \times Y) \cap \bigcap \rho & \text{rewrite } \lhd \text{ law 18.4} \\
\quad = \bigcap \{\ r : \rho \bullet (\bigcap \alpha \times Y) \cap r\ \} & \text{dist. } \cap - \bigcap \text{ law} \\
\quad = \bigcap \{\ r : \rho \bullet \bigcap \{ a : \alpha \bullet a \times Y\ \} \cap r\ \} & \text{dist. } \bigcup - \times \text{ law, } \alpha \neq \varnothing \\
\quad = \bigcap \{\ r : \rho \bullet \bigcap \{ a : \alpha \bullet (a \times Y) \cap r\ \}\ \} & \text{dist. } \bigcap - \cap \text{ law} \\
\quad = \bigcap \{\ r : \rho \bullet \bigcap \{ a : \alpha \bullet a \lhd r\ \}\ \} & \text{rewrite } \lhd \text{ law 18.4} \\
\quad = \bigcap \{\ a : \alpha;\ r : \rho \bullet a \lhd r\ \} & \text{flatten } \bigcup - \bigcup \text{ law}
\end{array}
$$

■

Proof of law 18.8, that $\_ \lhd \_$ pseudo-distributes through $\bigcup$

$$
\begin{array}{ll}
\bigcap \alpha \lhd \bigcup \rho & \\
\quad = \bigcup \rho \setminus (\bigcap \alpha \times Y) & \text{rewrite } \lhd \text{ law 18.4} \\
\quad = \bigcup \{\ r : \rho \bullet r \setminus (\bigcap \alpha \times Y) \cap r\ \} & \text{dist. } \setminus - \bigcup \text{ law} \\
\quad = \bigcup \{\ r : \rho \bullet r \setminus \bigcap \{ a : \alpha \bullet a \times Y\ \}\ \} & \text{dist. } \bigcap - \times \text{ law, } \alpha \neq \varnothing \\
\quad = \bigcup \{\ r : \rho \bullet \bigcup \{ a : \alpha \bullet r \setminus (a \times Y) \cap r\ \}\ \} & \text{p-dist. } \bigcup - \setminus \text{ law} \\
\quad = \bigcup \{\ r : \rho \bullet \bigcup \{ a : \alpha \bullet a \lhd r\ \}\ \} & \text{rewrite } \lhd \text{ law 18.4} \\
\quad = \bigcup \{\ a : \alpha;\ r : \rho \bullet a \lhd r\ \} & \text{flatten } \bigcup - \bigcup \text{ law}
\end{array}
$$

■

Proof of law 18.9, that $\_ \lhd \_$ pseudo-distributes through $\bigcap$

$$
\begin{array}{ll}
\bigcup \alpha_1 \lhd \bigcap \rho_1 & \\
\quad = \bigcap \rho_1 \setminus (\bigcup \alpha_1 \times Y) & \lhd \text{ as } \cap \text{ law 18.4} \\
\quad = \bigcap \{\ r : \rho_1 \bullet r \setminus (\bigcup \alpha_1 \times Y)\ \} & \text{dist. } \setminus\text{-}\bigcap \text{ law, } \rho_1 \neq \varnothing \\
\quad = \bigcap \{\ r : \rho_1 \bullet r \setminus \bigcup \{ a : \alpha_1 \bullet a \times Y\ \}\ \} & \text{dist. } \bigcup\text{-}\times \text{ law, } \alpha_1 \neq \varnothing \\
\quad = \bigcap \{\ r : \rho_1 \bullet \bigcap \{ a : \alpha_1 \bullet r \setminus (a \times Y)\ \}\ \} & \text{p-dist. } \bigcap\text{-}\setminus \text{ law} \\
\quad = \bigcap \{\ r : \rho_1 \bullet \bigcap \{ a : \alpha_1 \bullet a \lhd r\ \}\ \} & \lhd \text{ as } \cap \text{ law 18.4} \\
\quad = \bigcap \{\ a : \alpha_1;\ r : \rho_1 \bullet a \lhd r\ \} & \text{flatten } \bigcap\text{-}\bigcap \text{ law}
\end{array}
$$

■

Proof of law 18.11, that $\lhd$-distribution gives $\lhd$-distribution

$$
\begin{array}{ll}
a \lhd f(r, s) & \\
\quad = (X \setminus a) \lhd f(r, s) & \text{law 18.2, } b = X \\
\quad = f((X \setminus a) \lhd r, (X \setminus a) \lhd s) & \text{hyp., } a' = X \setminus a \\
\quad = f(a \lhd r, a \lhd s) & \text{law 18.2, } b = X
\end{array}
$$

■

Proof of law 18.12, that $a \lhd r = \varnothing \Leftrightarrow a \cap \operatorname{dom} r = \varnothing$

$$
\begin{array}{ll}
a \lhd r = \varnothing & \\
\quad \Leftrightarrow \{\ p : r \mid p.1 \in a\ \} = \{\ p : r \mid \mathit{false}\ \} & \text{defn. } \lhd, \varnothing \\
\quad \Leftrightarrow \{\ x : X;\ y : Y \mid x \mapsto y \in r \land x \in a\ \} = \{\ x : X;\ y : Y \mid \mathit{false}\ \} & \text{expand } p \\
\quad \Leftrightarrow (\ \forall x : X;\ y : Y \bullet x \mapsto y \in r \land x \in a \Leftrightarrow \mathit{false}\ ) & \text{equal set comp.} \\
\quad \Leftrightarrow (\ \forall x : X;\ y : Y \mid x \mapsto y \in r \land x \in a \bullet \mathit{false}\ ) & \text{simplify} \\
\quad \Leftrightarrow (\ \forall x : X \mid x \in \operatorname{dom} r \land x \in a \bullet \mathit{false}\ ) & \text{defn. dom} \\
\quad \Leftrightarrow (\ \forall x : X \mid x \in a \cap \operatorname{dom} r \bullet \mathit{false}\ ) & \text{defn. } \cap \\
\quad \Leftrightarrow a \cap \operatorname{dom} r = \varnothing & \text{defn. } \varnothing
\end{array}
$$

■

Proof of law 18.12, that $a \lhd r = r \Leftrightarrow \operatorname{dom} r \subseteq a$

$$
\begin{array}{ll}
a \lhd r = r & \\
\quad \Leftrightarrow \{\ p : r \mid p.1 \in a\ \} = \{\ p : r\ \} & \text{defn. } \lhd \\
\quad \Leftrightarrow \{\ x : X;\ y : Y \mid x \mapsto y \in r \land x \in a\ \} = \{\ x : X;\ y : Y \mid x \mapsto y \in r\ \} & \text{expand } p \\
\quad \Leftrightarrow (\ \forall x : X;\ y : Y \bullet x \mapsto y \in r \land x \in a \Leftrightarrow x \mapsto y \in r\ ) & \text{equal set comp.} \\
\quad \Leftrightarrow (\ \forall x : X;\ y : Y \mid x \mapsto y \in r \bullet x \in a\ ) & \text{simplify} \\
\quad \Leftrightarrow (\ \forall x : X \mid x \in \operatorname{dom} r \bullet x \in a\ ) & \text{defn. dom} \\
\quad \Leftrightarrow \operatorname{dom} r \subseteq a & \text{defn. } \subseteq
\end{array}
$$

■

Proof of law 18.14, that $a \lhd r = (a \cap \mathrm{dom}\, r) \lhd r$

$$
\begin{array}{ll}
(a \cap \mathrm{dom}\, r) \lhd r & \\
\quad = a \lhd r \cap \mathrm{dom}\, r \lhd r & \text{dist.} \\
\quad = a \lhd r \cap r & \text{law 18.13} \\
\quad = a \lhd r & \text{law 18.12}
\end{array}
$$

■

Proof of law 18.14, that $a \lhd\!\!\!- r = (a \cap \mathrm{dom}\, r) \lhd\!\!\!- r$

$$
\begin{array}{ll}
(a \cap \mathrm{dom}\, r) \lhd\!\!\!- r & \\
\quad = a \lhd\!\!\!- r \cup \mathrm{dom}\, r \lhd\!\!\!- r & \text{pseudo-dist.} \\
\quad = a \lhd\!\!\!- r \cup \varnothing & \text{law 18.13} \\
\quad = a \lhd\!\!\!- r &
\end{array}
$$

■

Proof of law 18.15, that restricting twice restricts to intersection

$$
\begin{array}{ll}
a \lhd b \lhd r & \\
\quad = (a \times Y) \cap (b \times Y) \cap r & \text{rewrite } \lhd \text{ law; assoc. } \cap \text{ law} \\
\quad = ((a \cap b) \times Y) \cap r & \text{dist. } \cap\text{-}\times \text{ law} \\
\quad = (a \cap b) \lhd r & \text{rewrite } \lhd \text{ law}
\end{array}
$$

■

Proof of law 19.4, that $\_(\!|\ \_\ |\!)$ distributes through $\bigcup$

$$
\begin{array}{ll}
\bigcup \rho (\!|\ \bigcup \alpha\ |\!) & \\
\quad = \mathrm{ran}(\bigcup \alpha \lhd \bigcup \rho) & \text{law 18.17} \\
\quad = \mathrm{ran}(\bigcup \{ a : \alpha;\ r : \rho \bullet a \lhd r\ \}) & \text{dist } \textstyle\bigcup\text{-}\lhd \text{ law} \\
\quad = \bigcup(\mathrm{ran}(\!|\ \{ a : \alpha;\ r : \rho \bullet a \lhd r\ \}\ |\!)) & \text{dist } \textstyle\bigcup\text{-ran law} \\
\quad = \bigcup \{ a : \alpha;\ r : \rho \bullet \mathrm{ran}(a \lhd r)\ \} & \text{law 19.3} \\
\quad = \bigcup \{ a : \alpha;\ r : \rho \bullet r(\!|\ a\ |\!)\ \} & \text{law 18.17}
\end{array}
$$

■

Proof of law 19.6, that lower image of a function distributes through $\cap$ on its set argument

$$
\begin{array}{ll}
\mathrm{lowerImage}\, f\ a \cap \mathrm{lowerImage}\, f\ b & \\
\quad = \{\ x : X \mid (\ \exists y : a \bullet x \mapsto y \in f\ ) \wedge (\ \exists y : b \bullet x \mapsto y \in f\ )\ \} & \text{defn. lowerImage, } \cap
\end{array}
$$

For any x in this set, the corresponding $y \in a$ and $y \in b$ must be the same y, because f is a function. Hence the y must be in $a \cap b$.

$$
\begin{array}{ll}
\quad = \{\ x : X \mid \exists y : a \cap b \bullet x \mapsto y \in f\ \} & \\
\quad = \mathrm{lowerImage}\, f (a \cap b) & \text{defn. lowerImage}
\end{array}
$$

■

Proof of law 19.7, that $a \cap \operatorname{dom} r \subseteq r^{\sim}(\!|\, r(\!|\, a \,|\!) \,|\!)$

$$
\begin{array}{ll}
r^{\sim}(\!|\, r(\!|\, a \,|\!) \,|\!) & \\
\quad = (r \fatsemi r^{\sim})(\!|\, a \,|\!) & \text{nested image law 20.31} \\
\quad \supseteq \operatorname{id}(\operatorname{dom} r)(\!|\, a \,|\!) & \text{id law 24.10; } \subset\text{-pres.} \\
\quad = \operatorname{ran} a \lhd \operatorname{id}(\operatorname{dom} r) & \_(\!|\, \_ \,|\!) \text{ law} \\
\quad = a \cap \operatorname{dom} r & \text{prop. id}
\end{array}
$$

■

Proof of law 19.8, that domain restriction becomes intersection

$$
\begin{array}{ll}
r(\!|\, a \cap b \,|\!) & \\
\quad = \operatorname{ran}((a \cap b) \lhd r) & \text{law 18.17} \\
\quad = \operatorname{ran}(a \lhd (b \lhd r)) & \text{law 18.15} \\
\quad = (b \lhd r)(\!|\, a \,|\!) & \text{law 18.17}
\end{array}
$$

■

Proof of law 19.9, that range restriction becomes intersection

$$
\begin{array}{ll}
(r \rhd b)(\!|\, a \,|\!) & \\
\quad = \operatorname{ran}(a \lhd (r \rhd b)) & \text{law 18.17} \\
\quad = \operatorname{ran}((a \lhd r) \rhd b) & \text{assoc. law 18.19} \\
\quad = \operatorname{ran}(a \lhd r) \cap b & \text{dual law 18.16} \\
\quad = r(\!|\, a \,|\!) \cap b & \text{law 18.17}
\end{array}
$$

■

Proof of law 19.10, that $a \lhd r \subseteq r \rhd r(\!|\, a \,|\!)$

$$
\begin{array}{ll}
r \rhd r(\!|\, a \,|\!) & \\
\quad = \{\ p : r \mid p.2 \in r(\!|\, a \,|\!)\ \} & \text{defn. } \rhd \\
\quad = \{\ p : r \mid p.2 \in \{\ q : r \mid q.1 \in a \bullet q.2\ \}\ \} & \text{defn. } \_(\!|\, \_ \,|\!) \\
\quad = \{\ p, q : r \mid q.1 \in a \wedge p.2 = q.2 \bullet p\ \} & \text{simplify} \\
\quad = \{\ p, q : r \mid q.1 \in a \wedge p.2 = q.2 \wedge (p.1 = q.1 \vee p.1 \neq q.1) \bullet p\ \} & \ldots \wedge \mathit{true} \\
\quad = \{\ p, q : r \mid q.1 \in a \wedge p = q \bullet p\ \} & \\
\qquad \cup \{\ p, q : r \mid q.1 \in a \wedge p.2 = q.2 \wedge p.1 \neq q.1 \bullet p\ \} & \text{defn. } \cup \\
\quad \supseteq \{\ p, q : r \mid q.1 \in a \wedge p = q \bullet p\ \} & \text{defn. } \subseteq \\
\quad = \{\ p : r \mid p.1 \in a\ \} & \text{simplify} \\
\quad = a \lhd r & \text{defn. } \lhd
\end{array}
$$

■

Proof of law 19.10, that $r \rhd r(\!|\ a\ |\!) \subseteq a \lhd r$

$$
\begin{array}{ll}
r \rhd r(\!|\ a\ |\!) & \\
\quad = r \setminus (r \rhd r(\!|\ a\ |\!)) & \text{partition law} \\
\quad \subseteq r \setminus (a \lhd r) & \setminus \text{ subset-order reversing law} \\
\quad = a \lhd r & \text{partition law}
\end{array}
$$

■

Proof of law 19.15, that upperBound pseudo-distributes through $\bigcap$:

$$
\begin{array}{ll}
\bigcap\{\ r : \rho;\ a : \alpha \bullet \text{upperBound}\, r\ a\ \} & \\
\quad = \bigcap\{\ r : \rho;\ a : \alpha \bullet \{\ y : Y \mid a \times \{y\} \subseteq r\ \}\ \} & \text{upperBound defn.} \\
\quad = \{\ y : Y \mid (\ \forall r : \rho;\ a : \alpha \bullet a \times \{y\} \subseteq r\ )\ \} & \bigcap\text{-simp. law} \\
\quad = \{\ y : Y \mid (\ \forall a : \alpha \bullet a \times \{y\} \subseteq \bigcap \rho\ )\ \} & \subseteq\text{-glb law 14.6} \\
\quad = \{\ y : Y \mid \bigcup \alpha \times \{y\} \subseteq \bigcap \rho\ \} & \subseteq\text{-lub law 14.5} \\
\quad = \text{upperBound}(\bigcap \rho)(\bigcup \alpha) & \text{upperBound defn.}
\end{array}
$$

■

Proof of law 19.17, that upper bound of a range restriction is intersection of the upperbound

$$
\begin{array}{ll}
\text{upperBound}(r \rhd b)a & \\
\quad = \{\ y : Y \mid a \times \{y\} \subseteq r \rhd b\ \} & \text{upperBound defn.} \\
\quad = \{\ y : Y \mid a \times \{y\} \subseteq r \cap (X \times b)\ \} & \rhd\text{-equiv. law} \\
\quad = \{\ y : Y \mid a \times \{y\} \subseteq r \wedge a \times \{y\} \subseteq X \times b\ \} & \subseteq\text{-}\cap\text{ law} \\
\quad = \{\ y : Y \mid a \times \{y\} \subseteq r \wedge \{y\} \subseteq b\ \} & \times\text{ law, } a \neq \varnothing \\
\quad = \{\ y : Y \mid a \times \{y\} \subseteq\ \} \cap b & \cap\text{ defn.} \\
\quad = (\text{upperBound}\, r\ a) \cap b & \text{upperBound defn.}
\end{array}
$$

■

Proof of law 19.18, that

$$
\begin{array}{ll}
\text{upperBound}(b \lhd r)a & \\
\quad = \{\ y : Y \mid b \times a \subseteq b \lhd r\ \} & \text{upperBound defn.} \\
\quad = \{\ y : Y \mid b \times a \subseteq (b \times Y) \cap r\ \} & \lhd\text{ equiv. dual law} \\
\quad = \{\ y : Y \mid b \times a \subseteq r \wedge a \subseteq b\ \} & \subseteq\text{-}\cap\text{ law; }\times\text{ law} \\
\quad = \{\ y : Y \mid b \times a \subseteq r\ \} \cap (\ \text{if } a \subseteq b \text{ then } Y \text{ else } \varnothing\ ) & \cap\text{ defn.; }\varnothing\text{ defn.} \\
\quad = (\ \text{if } a \subseteq b \text{ then upperBound}\, r\ a \text{ else } \varnothing\ ) & \text{upperBound defn.}
\end{array}
$$

■

Proof of law 19.20, that

$$
\begin{array}{ll}
\text{upperImage}((a \times b) \setminus r)c & \\
\quad = \{\ p : (a \times b) \setminus r \mid p.1 \in c \bullet p.2\ \} & \text{defn. upperImage} \\
\quad = \{\ y : Y \mid (\ \exists x : c \bullet x \mapsto y \in (a \times b) \setminus r\ )\ \} & \text{simplify} \\
\quad = \{\ y : Y \mid (\ \exists x : c \bullet x \in a \land y \in b \land x \mapsto y \notin r\ )\ \} & \text{defn. } \setminus, \times \\
\quad = \{\ y : b \mid (\ \exists x : c \cap a \bullet x \mapsto y \notin r\ )\ \} & \text{simplify} \\
\quad = \{\ y : b \mid (\ \exists x : c \bullet x \mapsto y \notin r\ )\ \} & \text{hyp. } c \subseteq a \\
\quad = \{\ y : b \mid \neg (\ \forall x : c \bullet x \mapsto y \in r\ )\ \} & \text{de Morgan} \\
\quad = b \setminus \{\ y : Y \mid (\ \forall x : c \bullet x \mapsto y \in r\ )\ \} & \text{defn. } \setminus \\
\quad = b \setminus (\text{upperBound}\ r\ c) & \text{law 19.14}
\end{array}
$$

■

Proof of law 20.9, that $\oplus$ distributes through $\bigcup$

$$
\begin{array}{ll}
\bigcup \rho \oplus s & \\
\quad = (\text{dom}\ s \lhd \bigcup \rho) \cup s & \text{defn. } \oplus \\
\quad = \bigcup \{\ r : \rho \bullet \text{dom}\ s \lhd r\ \} \cup s & \text{dist. } \lhd\text{-}\bigcup \text{ law 18.5} \\
\quad = \bigcup \{\ r : \rho \bullet (\text{dom}\ s \lhd r) \cup s\ \} & \cup \text{ distributes through } \bigcup \\
\quad = \bigcup \{\ r : \rho \bullet r \oplus s\ \} & \text{defn } \oplus
\end{array}
$$

■

Proof of law 20.11, that the domain of an overriding is the union of the separate domains:

$$
\begin{array}{ll}
\text{dom}(r \oplus s) & \\
\quad = \text{dom}((\text{dom}\ s \lhd r) \cup s) & \text{defn. } \oplus \\
\quad = \text{dom}(\text{dom}\ s \lhd r) \cup \text{dom}\ s & \text{dist. } \cup\text{-dom law} \\
\quad = (\text{dom}\ r \setminus \text{dom}\ s) \cup \text{dom}\ s & \lhd\text{-dom law} \\
\quad = \text{dom}\ r \cup \text{dom}\ s &
\end{array}
$$

■

Proof of law 20.12, that $\lhd$ distributes through $\oplus$:

$$
\begin{array}{ll}
a \lhd r \oplus a \lhd s & \\
\quad = \text{dom}(a \lhd s) \lhd (a \lhd r) \cup a \lhd s & \text{defn. } \oplus \\
\quad = (a \cap \text{dom}\ s) \lhd a \lhd r \cup a \lhd s & \text{law 18.16} \\
\quad = (a \setminus (a \cap \text{dom}\ s)) \lhd r \cup a \lhd s & \text{law 18.15} \\
\quad = (a \setminus \text{dom}\ s) \lhd r \cup a \lhd s & \text{closure} \\
\quad = a \lhd \text{dom}\ s \lhd r \cup a \lhd s & \text{law 18.15} \\
\quad = a \lhd (\text{dom}\ s \lhd r \cup s) & \text{law 18.5} \\
\quad = a \lhd (r \oplus s) & \text{defn. } \oplus
\end{array}
$$

■

Proof of law 20.13, that a restriction outside the domain of an overriding relation is independent of that relation.

$$
\begin{array}{ll}
a \lhd (r \oplus s) & \\
\quad = a \lhd r \oplus a \lhd s & \text{dist.} \\
\quad = a \lhd r \oplus (a \cap \operatorname{dom} s) \lhd s & \text{law 18.14} \\
\quad = a \lhd r \oplus \varnothing \lhd s & \text{hyp.} \\
\quad = a \lhd r \oplus \varnothing & \text{defn. } \lhd \\
\quad = a \lhd r & \text{law 20.7}
\end{array}
$$

■

Proof of law 20.13, that a restriction confined within the domain of an overriding relation depends on only that relation.

$$
\begin{array}{ll}
a \lhd (r \oplus s) & \\
\quad = a \lhd ((\operatorname{dom} s \lhd r) \cup s) & \text{defn. } \oplus \\
\quad = a \lhd \operatorname{dom} s \lhd r \cup a \lhd s & \cup\text{-}\oplus\text{-dist.} \\
\quad = (a \setminus \operatorname{dom} s) \lhd r \cup a \lhd s & \text{law 18.15} \\
\quad = \varnothing \lhd r \cup a \lhd s & \text{hyp.} \\
\quad = \varnothing \cup a \lhd s & \text{defn. } \lhd \\
\quad = a \lhd s &
\end{array}
$$

■

Proof of law 20.13, that a domain subtraction covering the domain of an overriding relation is independent of that relation.

$$
\begin{array}{ll}
a \lhd\!\!\!\lhd (r \oplus s) & \\
\quad = a \lhd\!\!\!\lhd r \oplus a \lhd\!\!\!\lhd s & \lhd\text{-dist, and law 18.11} \\
\quad = a \lhd\!\!\!\lhd r \oplus (a \cap \operatorname{dom} s) \lhd\!\!\!\lhd s & \text{law 18.14} \\
\quad = a \lhd\!\!\!\lhd r \oplus \operatorname{dom} s \lhd\!\!\!\lhd s & \text{hyp.} \\
\quad = a \lhd\!\!\!\lhd r \oplus \varnothing & \text{defn. } \lhd\!\!\!\lhd \\
\quad = a \lhd\!\!\!\lhd r & \text{law 20.7}
\end{array}
$$

■

Proof of law 20.15, that

$$
\begin{array}{ll}
(r \oplus s)(\!| \ a \ |\!) & \\
\quad = \operatorname{ran}(a \lhd (r \oplus s)) & \text{law 18.17} \\
\quad = \operatorname{ran}(a \lhd (\operatorname{dom} s \lhd\!\!\!\lhd r \cup s)) & \text{defn. } \oplus \\
\quad = \operatorname{ran}(a \lhd \operatorname{dom} s \lhd\!\!\!\lhd r \cup a \lhd s) & \lhd - \cup \text{ distbn.} \\
\quad = \operatorname{ran}(a \lhd \operatorname{dom} s \lhd\!\!\!\lhd r) \cup \operatorname{ran}(a \lhd s) & \operatorname{ran} - \cup \text{ distbn.} \\
\quad = \operatorname{ran}((a \setminus \operatorname{dom} s) \lhd r) \cup \operatorname{ran}(a \lhd s) & \text{law 18.15} \\
\quad = r(\!| \ a \setminus \operatorname{dom} s \ |\!) \cup s(\!| \ a \ |\!) & \text{law 18.17}
\end{array}
$$

■

Proof of law 20.21, that $\fatsemi$ distributes through $\bigcup$

$$
\begin{array}{ll}
\bigcup\{ \ r : \rho; \ s : \sigma \bullet r \fatsemi s \ \} & \\
\quad = \{ \ x : X; \ z : Z \mid (\ \exists r' : \{ \ r : \rho; \ s : \sigma \bullet r \fatsemi s \ \} \bullet x \mapsto z \in r' \ ) \ \} & \text{defn. } \bigcup \\
\quad = \{ \ x : X; \ z : Z \mid (\ \exists r' : X \leftrightarrow Z \bullet & \\
\qquad\qquad (\ \exists r : \rho; \ s : \sigma \bullet r' = r \fatsemi s \ ) & \\
\qquad\qquad \wedge \ x \mapsto z \in r' \ ) \ \} & \text{expand} \\
\quad = \{ \ x : X; \ z : Z \mid (\ \exists r : \rho; \ s : \sigma \bullet x \mapsto z \in r \fatsemi s \ ) \ \} & \text{elim. } r' \\
\quad = \{ \ x : X; \ z : Z \mid (\ \exists y : Y; \ r : \rho; \ s : \sigma \bullet x \mapsto y \in r \wedge y \mapsto z \in s \ ) \ \} & \text{defn. } \fatsemi \\
\quad = \{ \ x : X; \ z : Z \mid (\ \exists y : Y \bullet x \mapsto y \in \bigcup \rho \wedge y \mapsto z \in \bigcup \sigma \ ) \ \} & \text{defn. } \bigcup \\
\quad = \bigcup \rho \fatsemi \bigcup \sigma & \text{defn. } \fatsemi
\end{array}
$$

■

Proof of law 20.24, that

$$
\begin{array}{ll}
r \fatsemi s \cap \operatorname{id} X = \varnothing & \\
\quad \Leftrightarrow \{ \ x, x' : X; \ y : Y \mid x \mapsto y \in r \wedge y \mapsto x' \in s \bullet x \mapsto x' \ \} \cap \operatorname{id} X = \varnothing & \text{defn. } \fatsemi \\
\quad \Leftrightarrow \{ \ x : X; \ y : Y \mid x \mapsto y \in r \wedge y \mapsto x \in s \bullet x \mapsto x \ \} = \varnothing & \text{prop. id} \\
\quad \Leftrightarrow \{ \ x : X; \ y : Y \mid x \mapsto y \in r \wedge x \mapsto y \in s^{\sim} \bullet x \mapsto x \ \} = \varnothing & \text{rewrite} \\
\quad \Leftrightarrow r \cap s^{\sim} = \varnothing & \text{defn. } \cap
\end{array}
$$

■

Proof of law 20.25, that

$$
\begin{array}{ll}
\{ \ x : X; \ z : Z \mid \text{successors } r \ x \cap \text{predecessors } s \ z \} \neq \varnothing \ \} & \\
\quad = \{ \ x : X; \ z : Z \mid \{ y : Y \mid x \mapsto y \in r \wedge y \mapsto z \in s \ \} \neq \varnothing \ \} & \\
\qquad\qquad \text{defn. upper/lowerSingImage} & \\
\quad = \{ \ x : X; \ z : Z \mid \exists y : Y \bullet x \mapsto y \in r \wedge y \mapsto z \in s \ \} & \text{simplify} \\
\quad = r \fatsemi s & \text{defn. } \fatsemi
\end{array}
$$

■

Proof of law 20.26, that the range of a composition $r \fatsemi s$ is the upper image of the range of r through s.

$$
\begin{array}{ll}
\mathrm{ran}(r \fatsemi s) & \\
\quad = \mathrm{ran}\{\ x : X;\ z : Z \mid (\ \exists\, y : Y \bullet x \mapsto y \in r \land y \mapsto z \in s\ )\ \} & \text{defn. } \fatsemi \\
\quad = \{\ z : Z \mid (\ \exists\, x : X;\ y : Y \bullet x \mapsto y \in r \land y \mapsto z \in s\ )\ \} & \text{defn. ran} \\
\quad = \{\ z : Z \mid (\ \exists\, y : \mathrm{ran}\, r \bullet y \mapsto z \in s\ )\ \} & \text{defn. ran} \\
\quad = s(\!|\, \mathrm{ran}\, r \,|\!) & \text{defn. } \_(\!|\, \_ \,|\!)
\end{array}
$$

■

Proof of law 20.29, that $\lhd$ associates with $\fatsemi$

$$
\begin{array}{ll}
a \lhd (r \fatsemi s) & \\
\quad = \mathrm{id}\, a \fatsemi (r \fatsemi s) & \text{law 24.10} \\
\quad = (\mathrm{id}\, a \fatsemi r) \fatsemi s & \fatsemi\text{-assoc law 20.18} \\
\quad = (a \lhd r) \fatsemi s & \text{law 24.10}
\end{array}
$$

■

Proof of law 20.30, that $\rhd$ pseudo-distributes through $\fatsemi$

$$
\begin{array}{ll}
(r \rhd b) \fatsemi s & \\
\quad = (r \fatsemi \mathrm{id}\, b) \fatsemi s & \text{law 24.10} \\
\quad = r \fatsemi (\mathrm{id}\, b \fatsemi s) & \fatsemi \text{ assoc law 20.18} \\
\quad = r \fatsemi (b \lhd s) & \text{law 24.10}
\end{array}
$$

■

Proof of law 20.31, that nested images can be written as composition.

$$
\begin{array}{ll}
s(\!|\, r(\!|\, a \,|\!) \,|\!) & \\
\quad = s(\!|\, \mathrm{ran}(a \lhd r) \,|\!) & \text{law 18.17} \\
\quad = \mathrm{ran}((a \lhd r) \fatsemi s) & \text{law 20.27} \\
\quad = \mathrm{ran}(a \lhd (r \fatsemi s)) & \text{law 20.29} \\
\quad = (r \fatsemi s)(\!|\, a \,|\!) & \text{law 18.17}
\end{array}
$$

■

Proof of law 20.32, that overriding then composing on the right is a subset of the overriding of the individual compositions.

First, we prove the lemma that: $((\mathrm{dom}\, s) \lhd r \cup s) \fatsemi t \subseteq (\mathrm{dom}(s \fatsemi t) \lhd r \cup s) \fatsemi t$

$s \rhd \mathrm{dom}\, t \subseteq s$ — $r \rhd b \subseteq r$ (dual) law 18.12
$\Rightarrow \mathrm{dom}(s \rhd \mathrm{dom}\, t) \subseteq \mathrm{dom}\, s$ — dom $\subseteq$-order pres. law 17.4
$\Rightarrow \mathrm{dom}(s \fatsemi t) \subseteq \mathrm{dom}\, s$ — law 20.27
$\Rightarrow (\mathrm{dom}\, s) \lhd r \subseteq (\mathrm{dom}(s \fatsemi t)) \lhd r$ — $\lhd$ $\subseteq$-order rev.
$\Rightarrow (\mathrm{dom}\, s) \lhd r \cup s \subseteq (\mathrm{dom}(s \fatsemi t)) \lhd r \cup s$ — $\cup$ $\subseteq$-order pres.
$\Rightarrow ((\mathrm{dom}\, s) \lhd r \cup s) \fatsemi t \subseteq (\mathrm{dom}(s \fatsemi t) \lhd r \cup s) \fatsemi t$ — $\fatsemi$ $\subseteq$-order pres.

Then

$(r \oplus s) \fatsemi t$
$= ((\mathrm{dom}\, s) \lhd r \cup s) \fatsemi t$ — defn. $\oplus$
$\subseteq (\mathrm{dom}(s \fatsemi t) \lhd r \cup s) \fatsemi t$ — lemma
$= ((\mathrm{dom}(s \fatsemi t) \lhd r) \fatsemi t) \cup (s \fatsemi t)$ — $\fatsemi$-$\cup$ dist. law 20.21
$= (\mathrm{dom}(s \fatsemi t) \lhd (r \fatsemi t)) \cup (s \fatsemi t)$ — $\fatsemi$-$\lhd$ assoc. law 20.29
$= (r \fatsemi t) \oplus (s \fatsemi t)$ — defn. $\oplus$

■

Proof of law 20.34, that overriding then composing on the left is a superset of the overriding of the individual compositions.

$r \fatsemi (s \oplus t)$
$= r \fatsemi (((\mathrm{dom}\, t) \lhd s) \cup t)$ — defn. $\oplus$
$= r \fatsemi ((\mathrm{dom}\, t) \lhd s) \cup (r \fatsemi t)$ — $\fatsemi$-$\cup$ dist. law 20.21
$= ((r \rhd \mathrm{dom}\, t) \fatsemi s) \cup (r \fatsemi t)$ — pseudo-dist. law 20.30
$\supseteq (((\mathit{lowerImage}\ r\ \mathrm{dom}\, t) \lhd r) \fatsemi s) \cup (r \fatsemi t)$ — (dual) law 19.10
$= (((\mathrm{dom}(r \fatsemi t)) \lhd r) \fatsemi s) \cup (r \fatsemi t)$ — law 20.26
$= (\mathrm{dom}(r \fatsemi t) \lhd (r \fatsemi s)) \cup (r \fatsemi t)$ — $\fatsemi$-$\lhd$ assoc. law 20.29
$= (r \fatsemi s) \oplus (r \fatsemi t)$ — defn. $\oplus$

■

Proof of law 20.37, that $\overrightarrow{\fatsemi}$ pseudo-distributes through $\cup$ on its first argument

$$
\begin{array}{ll}
(r \cup r') \overrightarrow{\fatsemi} s & \\
\quad = \{\ x : X;\ z : Z \mid (\ \forall y : Y \mid x \mapsto y \in (r \cup r') \bullet y \mapsto z \in s\ )\ \} & \text{defn. } \overrightarrow{\fatsemi} \\
\quad = \{\ x : X;\ z : Z \mid (\ \forall y : Y \bullet x \mapsto y \in (r \cup r') \Rightarrow y \mapsto z \in s\ )\ \} & \text{defn. } \forall \\
\quad = \{\ x : X;\ z : Z \mid (\ \forall y : Y \bullet x \mapsto y \in r \lor x \mapsto y \in r' \Rightarrow y \mapsto z \in s\ )\ \} & \text{defn. } \cup \\
\quad = \{\ x : X;\ z : Z \mid (\ \forall y : Y \bullet (x \mapsto y \in r \Rightarrow y \mapsto z \in s) & \\
\qquad\qquad \land (x \mapsto y \in r' \Rightarrow y \mapsto z \in s)\ )\ \} & \text{rearrange} \\
\quad = \{\ x : X;\ z : Z \mid (\ \forall y : Y \bullet x \mapsto y \in r \Rightarrow y \mapsto z \in s\ ) & \\
\qquad\qquad \land (\ \forall y : Y \bullet x \mapsto y \in r' \Rightarrow y \mapsto z \in s\ )\ \} & \text{prop. } \forall \\
\quad = \{\ x : X;\ z : Z \mid (\ \forall y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in s\ )\ \} & \\
\qquad\qquad \cap \{\ x : X;\ z : Z \mid (\ \forall y : Y \mid x \mapsto y \in r' \bullet y \mapsto z \in s\ )\ \} & \text{defn. } \cap \\
\quad = r \overrightarrow{\fatsemi} s \cap r' \overrightarrow{\fatsemi} s & \text{defn. } \overrightarrow{\fatsemi}
\end{array}
$$

■

Proof of law 20.39, that

$$
\begin{array}{ll}
\{\ x : X;\ z : Z \mid \text{successors}\ r\ x \subseteq \text{predecessors}\ s\ z\ \} & \\
\quad = \{\ x : X;\ z : Z \mid \{\ y : Y \mid x \mapsto y \in r\ \} \subseteq \{y : Y \mid y \mapsto z \in s\ \}\ \} & \\
\qquad\qquad \text{defn. upper/lowerSingImage} & \\
\quad = \{\ x : X;\ z : Z \mid \forall y : \{\ y' : Y \mid x \mapsto y' \in r\ \} \bullet y \in \{y' : Y \mid y' \mapsto z \in s\ \}\ \} & \text{law } \subseteq \\
\quad = \{\ x : X;\ z : Z \mid \forall y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in s\ \} & \text{simplify} \\
\quad = r \overrightarrow{\fatsemi} s & \text{defn. } \overrightarrow{\fatsemi}
\end{array}
$$

■

Proof of law 20.40, that right demonic composition always includes the complement of the domain

$$
\begin{array}{ll}
r \overrightarrow{\fatsemi} s & \\
\quad = \mathit{dom} r \lhd (r \overrightarrow{\fatsemi} s) \cup \text{dom}\ r \lhd (r \overrightarrow{\fatsemi} s) & \text{partition law} \\
\quad = \mathit{dom} r \lhd \{\ x : X;\ z : Z \mid \forall y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in s\ \} & \\
\qquad\qquad \cup \text{dom}\ r \lhd (r \overrightarrow{\fatsemi} s) & \text{defn. } \overrightarrow{\fatsemi} \\
\quad = \{\ x : X \setminus \text{dom}\ r;\ z : Z \mid \forall y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in s\ \} & \\
\qquad\qquad \cup \text{dom}\ r \lhd (r \overrightarrow{\fatsemi} s) & \text{defn. } \lhd \\
\quad = \{\ x : X \setminus \text{dom}\ r;\ z : Z \mid \mathit{true}\ \} \cup \text{dom}\ r \lhd (r \overrightarrow{\fatsemi} s) & \text{empty } \forall \\
\quad = ((X \setminus \text{dom}\ r) \times Z) \cup \text{dom}\ r \lhd (r \overrightarrow{\fatsemi} s) & \text{simplify}
\end{array}
$$

■

Proof of law 20.41, that functional right demonic composition reduces to composition

$$
\begin{array}{ll}
f \overset{\rightarrow}{\fatsemi} s & \\
\quad = ((X \setminus \operatorname{dom} f) \times Z & \text{law 20.40; defn } \overset{\rightarrow}{\fatsemi} \\
\qquad \cup \{\ x : \operatorname{dom} r;\ z : Z \mid \forall\, y : Y \mid x \mapsto y \in f \bullet y \mapsto z \in s\ \} & \\
\quad = ((X \setminus \operatorname{dom} f) \times Z & \\
\qquad \cup \{\ x : \operatorname{dom} r;\ z : Z \mid \forall\, y : Y \mid y = f\ x \bullet y \mapsto z \in s\ \} & \text{functional } f \\
\quad = ((X \setminus \operatorname{dom} f) \times Z & \\
\qquad \cup \{\ x : \operatorname{dom} r;\ z : Z \mid \exists\, y : Y \mid y = f\ x \bullet y \mapsto z \in s\ \} & \text{one point} \\
\quad = ((X \setminus \operatorname{dom} f) \times Z \cup (f \fatsemi s) & \text{defn. } \fatsemi
\end{array}
$$

■

Proof of law 20.42, that right demonic composition with an inverse function reduces to composition

$$
\begin{array}{ll}
r \overset{\rightarrow}{\fatsemi} s & \\
\quad = ((X \setminus \operatorname{dom} r) \times Z & \text{law 20.40; defn } \overset{\rightarrow}{\fatsemi} \\
\qquad \cup \{\ x : \operatorname{dom} r;\ z : Z \mid \forall\, y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in s\ \} &
\end{array}
$$

For every z in the set, there is only one $y \in \operatorname{dom} s$ that maps to it (because s is inverse functional). So there can be only one $x \in \operatorname{dom} r$ mapping to that y. So r must be functional at that x.

$$
\begin{array}{ll}
\quad = ((X \setminus \operatorname{dom} r) \times Z & \\
\qquad \cup \{\ x : \operatorname{dom} r;\ z : Z \mid r \operatorname{functionalAt} x & \\
\qquad\qquad \wedge (\ \exists\, y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in s\ )\ \} & \\
\quad = ((X \setminus \operatorname{dom} r) \times Z & \text{defn. } \fatsemi \\
\qquad \cup \{\ x : \operatorname{dom} r;\ z : Z \mid r \operatorname{functionalAt} x \wedge (x \mapsto z \in r \fatsemi s)\ \} & \\
\quad = ((X \setminus \operatorname{dom} r) \times Z & \text{defn. } \cap \\
\qquad \cup (\{\ x : \operatorname{dom} r;\ z : Z \mid r \operatorname{functionalAt} x\ \} \cap \operatorname{dom} r \lhd (r \fatsemi s)) & \\
\quad = ((X \setminus \operatorname{dom} r) \times Z & \text{rearrange} \\
\qquad \cup \{\ x : \operatorname{dom} r \mid r \operatorname{functionalAt} x\ \} \lhd (r \fatsemi s) &
\end{array}
$$

■

Proof of law 20.43, that a de Morgan style law relates right demonic composition and composition

$$
\begin{array}{ll}
(X \times Z) \setminus (r \fatsemi ((Y \times Z) \setminus s)) & \\
\quad = (X \times Z) \setminus \{\ x : X;\ z : Z \mid \exists\, y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in (Y \times Z) \setminus s\ \} & \text{defn } \fatsemi \\
\quad = (X \times Z) \setminus \{\ x : X;\ z : Z \mid \exists\, y : Y \mid x \mapsto y \in r \bullet y \mapsto z \notin s\ \} & \text{simplify} \\
\quad = (X \times Z) \setminus \{\ x : X;\ z : Z \mid \neg\ (\ \forall\, y : Y \mid x \mapsto y \in r \bullet y \mapsto z \in s\ )\ \} & \exists \equiv \neg\ \forall \neg \\
\quad = (X \times Z) \setminus ((X \times Z) \setminus (r \overset{\rightarrow}{\fatsemi} s)) & \text{defn } \overset{\rightarrow}{\fatsemi} \\
\quad = (X \times Z) \cap (r \overset{\rightarrow}{\fatsemi} s) & \text{law } \setminus \\
\quad = r \overset{\rightarrow}{\fatsemi} s & \text{simplify}
\end{array}
$$

■

Proof of law 21.5, that the function constructor distributes through non-empty generalised intersection.

$$
\begin{array}{lr}
\bigcap \alpha \nrightarrow \bigcap \beta & \\
\quad = \{\ f : X \leftrightarrow Y \mid f \in \bigcap \alpha \leftrightarrow \bigcap \beta \wedge (\ \forall p, q : f \mid p.1 = q.1 \bullet p = q\ )\ \} & \text{defn. } \nrightarrow \\
\quad = \{\ f : X \leftrightarrow Y \mid f \in \bigcap \{\ a : \alpha;\ b : \beta \bullet a \leftrightarrow b\ \} & \\
\qquad\qquad \wedge (\ \forall p, q : f \mid p.1 = q.1 \bullet p = q\ )\ \} & \text{property } \leftrightarrow \\
\quad = \{\ f : X \leftrightarrow Y \mid (\ \forall a : \alpha;\ b : \beta \bullet f \in a \leftrightarrow b\ ) & \\
\qquad\qquad \wedge (\ \forall p, q : f \mid p.1 = q.1 \bullet p = q\ )\ \} & \text{simplify } \bigcup \\
\quad = \{\ f : X \leftrightarrow Y \mid (\ \forall a : \alpha;\ b : \beta \bullet & \\
\qquad\qquad f \in a \leftrightarrow b & \\
\qquad\qquad \wedge (\ \forall p, q : f \mid p.1 = q.1 \bullet p = q\ )\ )\ \} & \alpha, \beta \neq \varnothing \\
\quad = \{\ f : X \leftrightarrow Y \mid (\ \forall a : \alpha;\ b : \beta \bullet f \in a \nrightarrow b\ )\ \} & \text{defn } \nrightarrow \\
\quad = \bigcap \{\ a : \alpha;\ b : \beta \bullet a \nrightarrow b\ \} & \text{simplify } \bigcup
\end{array}
$$

■

Proof of law 21.11, that a relation is a function precisely when the composition of its inverse with itself yields the identity.

$$
\begin{array}{lr}
r^{\sim} \fatsemi r = \mathrm{id}(\mathrm{ran}\, r) & \\
\quad \Leftrightarrow \{\ y, y' : Y \mid (\ \exists x : X \bullet y \mapsto x \in r^{\sim} \wedge x \mapsto y' \in r\ )\ \} & \text{defn. } \fatsemi \\
\qquad = \{\ y : \mathrm{ran}\, r \bullet y \mapsto y\ \} & \text{defn. id} \\
\quad \Leftrightarrow \{\ y, y' : Y \mid (\ \exists x : X \bullet x \mapsto y \in r \wedge x \mapsto y' \in r\ )\ \} & \text{prop. } {}^{\sim} \\
\qquad = \{\ y : Y \mid (\ \exists x : X \bullet x \mapsto y \in r\ ) \bullet y \mapsto y\ \} & \text{defn. ran} \\
\quad \Leftrightarrow \{\ y, y' : Y \mid (\ \exists x : X \bullet x \mapsto y \in r \wedge x \mapsto y' \in r\ )\ \} & \\
\qquad = \{\ y, y' : Y \mid (\ \exists x : X \bullet x \mapsto y \in r \wedge x \mapsto y' \in r\ ) \wedge y = y'\ \} & \text{rearranging} \\
\quad \Leftrightarrow (\ \forall y, y' : Y \bullet (\ \exists x : X \bullet x \mapsto y \in r \wedge x \mapsto y' \in r\ ) & \\
\qquad \Leftrightarrow (\ \exists x : X \bullet x \mapsto y \in r \wedge x \mapsto y' \in r\ ) \wedge y = y'\ ) & \text{set equality} \\
\quad \Leftrightarrow (\ \forall y, y' : Y \bullet (\ \exists x : X \bullet x \mapsto y \in r \wedge x \mapsto y' \in r\ ) \Leftrightarrow y = y'\ ) & \text{simplifying} \\
\quad \Leftrightarrow r \in X \nrightarrow Y & \text{defn. } \nrightarrow
\end{array}
$$

■

Proof of law 21.14, that unequal source sets give disjoint total functions:

$$
\begin{array}{lr}
(a \rightarrow Y) \cap (a' \rightarrow Y) & \\
\quad = \{\ f : a \nrightarrow Y \mid \mathrm{dom}\, f = a\ \} \cap \{\ f : a' \nrightarrow Y \mid \mathrm{dom}\, f = a'\ \} & \text{defn } \rightarrow \\
\quad = \varnothing & \text{hyp. } a \neq a'
\end{array}
$$

■

Proof of law 21.15, that $\rightarrow$ distributes through $\bigcap$ on the left:

$$
\begin{array}{ll}
X \rightarrow \bigcap \beta & \\
\quad = \{\ f : X \pfun Y \mid f \in X \pfun \bigcap \beta \land \operatorname{dom} f = X\ \} & \text{defn. } \rightarrow \\
\quad = \{\ f : X \pfun Y \mid f \in \bigcap \{\ b : \beta \bullet X \pfun b\ \} \land \operatorname{dom} f = X\ \} & \text{law 21.5, } \beta \neq \varnothing \\
\quad = \{\ f : X \pfun Y \mid (\ \forall\, b : \beta \bullet f \in X \pfun b\ ) \land \operatorname{dom} f = X\ \} & \text{simplify } \bigcup \\
\quad = \{\ f : X \pfun Y \mid (\ \forall\, b : \beta \bullet f \in X \pfun b \land \operatorname{dom} f = X\ )\ \} & \beta \neq \varnothing \\
\quad = \{\ f : X \pfun Y \mid (\ \forall\, b : \beta \bullet f \in X \rightarrow b\ )\ \} & \text{defn. } \rightarrow \\
\quad = \bigcap \{\ b : \beta \bullet X \rightarrow b\ \} & \text{simplify } \bigcup
\end{array}
$$

■

Proof of law 21.26, that of the lack of surjections.

We proceed by *reductio ad absurdum*; by assuming the contrary and showing it leads to a contradiction.

Suppose the contrary: $X \twoheadrightarrow \mathbb{P}\, X \neq \varnothing$. So there is at least one surjection, call it $f : X \twoheadrightarrow \mathbb{P}\, X$

Applying f to any element $x : X$ in its domain results in a set of elements of X, which set may or may not include x. Let ξ be the set of all xs where $f\ x$ does *not* include x.

$$\xi == \{\ x : \operatorname{dom} f \mid x \notin f\ x\ \}$$

From its definition, we have $\xi \in \mathbb{P}\, X$. Since f is a surjection, we have $\operatorname{ran} f = \mathbb{P}\, X$. Hence $\xi \in \operatorname{ran} f$. Since f is a surjection, there must be a domain element that maps to ξ.

$$\exists\, x : \operatorname{dom} f \bullet f\ x = \xi$$

Let us call it x_ξ

$$
\begin{array}{|l}
x_\xi : \operatorname{dom} f \\
\hline
f\ x_\xi = \xi
\end{array}
$$

From the definition of ξ we have $\forall\, x : \xi \bullet x \notin f\ x$

So if $x_\xi \in \xi$ then it is not in $f\ x_\xi$, so $x_\xi \in \xi \Rightarrow x_\xi \notin f\ x_\xi$

But the definition of x_ξ gives us $f\ x_\xi = \xi$. Hence $x_\xi \in \xi \Rightarrow x_\xi \notin \xi$. This tells us $x_\xi \notin \xi$

Returning to the definition $\xi = f\ x_\xi$, we have $x_\xi \notin \xi \Leftrightarrow x_\xi \notin f\ x_\xi$

If x_ξ is not in $f\ x_\xi$, it satisfies the definition of ξ, so $x_\xi \notin f\ x_\xi \Rightarrow x_\xi \in \xi$

Hence $x_\xi \in \xi$

From our assumption we have deduced $x_\xi \notin \xi \land x_\xi \in \xi$, which is *false*. Whence we conclude the negation of that assumption:

$$X \twoheadrightarrow \mathbb{P}\, X = \varnothing$$

■

Proof of law 23.1, that a monoid has a unique identity element

$$
\begin{array}{ll}
e_0 & \\
\quad = e \diamond e_0 & \text{defn. } e \\
\quad = e & \text{hyp}[e/x]
\end{array}
$$

■

Proof of law 23.2, that a monoid's binary function is a surjection.

Follows directly from the definition of e.

■

Proof of law 23.3, that a group has a unique inverse function

$$
\begin{array}{ll}
inv_0\ x & \\
\quad = inv_0\ x \diamond e & \text{defn. } e \\
\quad = inv_0\ x \diamond\ (x \diamond inv\ x) & \text{defn. } inv \\
\quad = (inv_0\ x \diamond x) \diamond inv\ x & \text{assoc. } \diamond \\
\quad = e \diamond inv\ x & \text{hyp.} \\
\quad = inv\ x & \text{defn. } e
\end{array}
$$

■

Proof of law 23.4, that multiplication modulo a prime forms an abelian group:

We must prove the following properties:

- (*SemiGroup*) The binary operation modulo multiplication is total on g:
  Modulo multiplication is total because multiplication is total, and modulus is total.
- (*SemiGroup*) The binary operation is closed (range is a subset of g):
  From the definition of *mod*, we know that

  $$(n * m) \bmod p \in 0 \,..\, p - 1$$

  So we need to show

  $$p : \text{prime} \vdash \forall\, n, m : 1 \,..\, p - 1 \bullet (n * m) \bmod p \neq 0$$

  This is obviously true for $n, m = 1$, so assume $n, m > 1$. $(n * m) \bmod p = 0$ means that p divides $n * m$, which requires n or m to be a (non-unit) factor of p. But p is prime, and so has no such factors.
- (*SemiGroup*) The binary operation is associative:
  Modulo multiplication is associative because multiplication is associative, and modulus forms a homomorphism (law 30.12).
- (*Abelian*) The binary operation is commutative:
  Modulo multiplication is commutative because multiplication is commutative, and modulus forms a homomorphism (law 30.12).

- (*Monoid*) The binary operation has a unit in g:
  The unit is 1:

  $$p : \text{prime} \vdash \forall\, n : 1 \mathinner{..} p - 1 \bullet (1 * n) \bmod p = n = (n * 1) \bmod p$$

- (*Group*) The binary operation has an inverse:
  First we show the operation is surjective (and hence, if it has an inverse function, that inverse is total):

  $$p : \text{prime} \vdash \forall\, n, n' : 1 \mathinner{..} p - 1 \bullet \exists\, m : 1 \mathinner{..} p - 1 \bullet (n * m) \bmod p = n'$$

  We use the well-known result, derived from Euclid's algorithm for highest common factor, that if two numbers n and p have a highest common factor of h, then

  $$\exists\, a, b : \mathbb{Z} \bullet n * a + p * b = h$$

  Here $h = 1$. Multiply both sides by n' to get

  $$n * a * n' + p * b * n' = n'$$

  Take the modulus (and use $n' < p$):

  $$(n * a * n') \bmod p = n'$$

  Hence $m = (a * n') \bmod p$.
  Next, since p is finite, the *pigeonhole principle*, law 15.7, gives us that the operation is injective, and hence has a (total) inverse, as required.

■

Proof of law 24.14, that $\text{id}(\text{dom}\, r) \subseteq r \fatsemi r^\sim$

$$\begin{array}{lr}
r \fatsemi r^\sim & \\
= \{\ x, x' : X;\ y, y' : Y \mid x \mapsto y \in r \land y' \mapsto x' \in r^\sim \mid y = y' \bullet x \mapsto x'\ \} & \text{defn. } \fatsemi \\
= \{\ x, x' : X;\ y : Y \mid x \mapsto y \in r \land x' \mapsto y \in r \bullet x \mapsto x'\ \} & \text{rewriting} \\
\supseteq \{\ x : X;\ y : Y \mid x \mapsto y \in r \bullet x \mapsto x\ \} & \text{defn. } \subseteq \\
= \{\ x : \text{dom}\, r \bullet x \mapsto x\ \} & \text{defn. dom} \\
= \text{id}(\text{dom}\, r) & \text{defn. id}
\end{array}$$

■

Proof of law 24.18, that a composition of two relations is irreflexive precisely when the cyclic permutation of the composition is irreflexive

$$\begin{array}{lr}
r \fatsemi s \in \text{irreflexive}\, X & \\
\Leftrightarrow r \fatsemi s \cap \text{id}\, X = \varnothing & \text{defn. } \textit{irreflexive} \\
\Leftrightarrow r \cap s^\sim = \varnothing & \text{disjointness law 20.24} \\
\Leftrightarrow (r \cap s^\sim)^\sim = \varnothing^\sim & \text{invert} \\
\Leftrightarrow r^\sim \cap s = \varnothing & \text{order pres. law 16.7; } s^{\sim\sim} = s;\ \varnothing^\sim = \varnothing \\
\Leftrightarrow s \fatsemi r \cap \text{id}\, Y = \varnothing & \cap \text{ commutative law 13.20; disjoint law 20.24} \\
\Leftrightarrow s \fatsemi r \in \text{irreflexive}\, Y & \text{defn. } \textit{irreflexive}
\end{array}$$

■

Proof of law 24.22, that the composition of any relation with its inverse is symmetric.

$$
\begin{array}{ll}
(r \fatsemi r^\sim)^\sim & \\
\quad = r^{\sim\sim} \fatsemi r^\sim & \text{law 20.23 with } s = r^\sim \\
\quad = r \fatsemi r^\sim & \text{law 16.5}
\end{array}
$$

■

Proof of law 24.29, that any restriction of a transitive relation is transitive

$$
\begin{array}{ll}
(a \lhd r \rhd b) \fatsemi (a \lhd r \rhd b) & \\
\quad \subseteq a \lhd r \fatsemi r \rhd b & r \subseteq b;\ \fatsemi \text{ subset order preserving} \\
\quad \subseteq a \lhd r \rhd b & \text{hyp; } \lhd \text{ subset order preserving}
\end{array}
$$

■

Proof of law 24.29, that the intersection of two transitive relations is transitive

$$
\begin{array}{ll}
(r \cap s) \fatsemi (r \cap s) & \\
\quad \subseteq r \fatsemi r \cap r \fatsemi s \cap s \fatsemi r \cap s \fatsemi s & \fatsemi - \cap \text{ law} \\
\quad \subseteq r \cap s \cap r \fatsemi s \cap s \fatsemi r & \text{hyp.} \\
\quad \subseteq r \cap s & \text{prop. } \cap
\end{array}
$$

■

Proof of law 24.30, that a transitive, symmetric relation is reflexive on its domain:

$$
\begin{array}{ll}
x \in \mathrm{dom}\, r & \\
\quad \Rightarrow x \mapsto x' \in r & \\
\quad \Rightarrow x' \mapsto x \in r & \text{hyp. symmetric} \\
\quad \Rightarrow x \mapsto x \in r & \text{hyp. transitive} \\
\quad \Rightarrow \mathrm{id}(\mathrm{dom}\, r) \subseteq r &
\end{array}
$$

■

Proof of law 24.32, that any transitive function is the identity on its range

$$
\begin{array}{l}
\text{transitive } X \Rightarrow f \fatsemi f \subseteq f \\
\text{transitive } X \cap X \nrightarrow X \\
\quad \Rightarrow (\ \forall x : \mathrm{dom}\, f \mid f\ x \in \mathrm{dom}\, f \bullet f(f\ x) = f\ x\ ) \\
\quad \Rightarrow (\ \forall x : \mathrm{dom}\, f;\ y : X \mid y = f\ x \in \mathrm{dom}\, f \bullet f\ y = y\ ) \\
\quad \Rightarrow (\ \forall y : X \mid y \in \mathrm{ran}\, f \bullet f\ y = y\ ) \\
\quad \Rightarrow \mathrm{ran}\, f \lhd f = \mathrm{id}(\mathrm{ran}\, f)
\end{array}
$$

■

Proof of law 24.33, that a transitive, irreflexive function has no interior vertices:

Let $y = \operatorname{dom} r \cap \operatorname{ran} r$ be an interior point. Then $\exists\, x, z : X \bullet \{x \mapsto y, y \mapsto z\} \subseteq r$

By transitivity, $x \mapsto y \in r$. By functionality, $y = z$.

This contradicts irreflexivity, hence no such y exists.

■

Proof of law 24.38, that $\_^+$ is $\subseteq$-order preserving:

Follows from the alternative definition in terms of relational iteration law 24.37, from the order-preserving property of relational iteration law 31.5, and from the order-preserving property of set union law 13.10.

■

Proof of law 24.40, that the transitive closure of a relation is irreflexive precisely when all the iterates of that relation are irreflexive.

$$
\begin{array}{lr}
r^+ \in \text{irreflexive } X & \\
\quad \Leftrightarrow \bigcup\{\ n : \mathbb{N}_1 \bullet r^n\ \} \cap \operatorname{id} X = \varnothing & \text{law 24.37; defn. } \textit{irreflexive} \\
\quad \Leftrightarrow \bigcup\{\ n : \mathbb{N}_1 \bullet r^n \cap \operatorname{id} X\ \} = \varnothing & \text{distributive law 13.9} \\
\quad \Leftrightarrow (\ \forall\, n : \mathbb{N}_1 \bullet r^n \cap \operatorname{id} X = \varnothing\ ) & \text{law 13.18} \\
\quad \Leftrightarrow (\ \forall\, n : \mathbb{N}_1 \bullet r^n \in \text{irreflexive } X\ ) & \text{defn. } \textit{irreflexive}
\end{array}
$$

■

Proof of law 26.3, that $r \setminus \operatorname{id} a \in \operatorname{order} X$.

Subsetting preserves antisymmetry, so we are left with proving transitivity. That is:

$$
\begin{array}{lr}
x \mapsto y \in r \setminus \operatorname{id} a \wedge y \mapsto z \in r \setminus \operatorname{id} a \Rightarrow x \mapsto z \in r \setminus \operatorname{id} a & \\
\quad x \mapsto y \in r \wedge x \mapsto y \notin \operatorname{id} a \wedge y \mapsto z \in r \wedge y \mapsto z \notin \operatorname{id} a \Rightarrow x \mapsto z \notin \operatorname{id} a & r \text{ transitive} \\
\quad x \mapsto z \in \operatorname{id} a \Rightarrow x \mapsto y \notin r \vee x \mapsto y \in \operatorname{id} a \vee y \mapsto z \notin r \vee y \mapsto z \in \operatorname{id} a & \Rightarrow \text{identity} \\
\quad x \in a \Rightarrow x \mapsto y \notin r \vee x \mapsto y \in \operatorname{id} a \vee y \mapsto x \notin r \vee y \mapsto x \in \operatorname{id} a & \text{simplify id} \\
\quad x \in a \Rightarrow x \mapsto y \notin (r \cap r^{\sim}) \vee x = y \in a & \text{defn. } {}^{\sim}\text{; simplify} \\
\quad x \in a \Rightarrow x = y \notin \operatorname{dom}(r \cap r^{\sim}) \vee x \neq y \vee x = y \in a & r \text{ antisymmetric} \\
\quad \textit{true} & \text{simplify}
\end{array}
$$

■

Proof of law 26.4, that transitive, irreflexive relations are irreflexive orders.

This is an immediate consequence of law 24.56 and the definition of irreflexive order.

■

Proof of law 26.36, that graph-preserving map applied to *glb* gives wider bounds.

The definition of *glb* gives us that it precedes every element in a set, so

$$
\begin{array}{ll}
(\ \forall x' : a \bullet glb\ r_x\ a \mapsto x' \in r_x\ ) & \text{defn. } glb \\
\quad \Rightarrow (\ \forall x' : a \cap \mathrm{dom} f \bullet f(glb\ r\ a) \mapsto f\ x' \in r_y\ ) & \text{hyp. graph preserving} \\
\quad \Rightarrow f(glb\ r\ a) \mapsto glb\ r_y(f(\!|\ a\ |\!)) \in r_y & \text{law 26.28}
\end{array}
$$

■

Proof of law 26.37, that any function that distributes through least upper bound or greatest lower bound is graph-preserving

From the hypothesis, choose the case where $\beta = \{x, y\}$. So

$$
\begin{array}{ll}
x \mapsto y \in r_x \Rightarrow y = \mathrm{lub}\, r_x\{x, y\} & \text{prop. } lub \\
\wedge\ f(\mathrm{lub}\, r_x\ \alpha) = \mathrm{lub}\, r_y(f(\!|\ \alpha\ |\!)) & \text{hyp.} \\
x \mapsto y \in r_x \Rightarrow y = \mathrm{lub}\, r_x\{x, y\} & \\
\wedge\ f(\mathrm{lub}\, r_x\{x, y\}) = \mathrm{lub}\, r_y\{f\ x, f\ y\} & \text{choice of } \alpha \\
x \mapsto y \in r_x \Rightarrow f\ y = \mathrm{lub}\, r_y\{f\ x, f\ y\} & \text{rearrange} \\
x \mapsto y \in r_x \Rightarrow f\ x \mapsto f\ y \in r_y & \text{prop. } lub
\end{array}
$$

■

Proof of law 28.1, that the additive identity element acts as a zero for multiplication

$$
\begin{array}{ll}
x \dot{*} O = x \dot{*} (O \dot{+} O) & \text{defn. } O \\
\quad \Rightarrow x \dot{*} O = x \dot{*} O \dot{+} x \dot{*} O & \text{distributive} \\
\quad \Rightarrow x \dot{*} O \dot{+} \dot{-}(x \dot{*} O) = (x \dot{*} O \dot{+} x \dot{*} O) \dot{+} \dot{-}(x \dot{*} O) & \\
\quad \Rightarrow x \dot{*} O \dot{+} \dot{-}(x \dot{*} O) = x \dot{*} O \dot{+} (x \dot{*} O \dot{+} \dot{-}(x \dot{*} O)) & \text{assoc.} \\
\quad \Rightarrow O = x \dot{*} O \dot{+} O & \text{defn. } \dot{-} \\
\quad \Rightarrow O = x \dot{*} O & \text{defn. } O
\end{array}
$$

■

Proof of law 28.2, that the additive inverse distributes through multiplication

$$
\begin{array}{ll}
O = x \dot{*} O & \text{ring zero law 28.1} \\
\quad \Rightarrow O = x \dot{*} (y \dot{+} \dot{-} y) & \text{defn. } \dot{-} \\
\quad \Rightarrow O = x \dot{*} y \dot{+} x \dot{*} \dot{-} y & \text{distributive} \\
\quad \Rightarrow \dot{-}(x \dot{*} y) = x \dot{*} \dot{-} y & \text{defn } \dot{-}
\end{array}
$$

■

Proof of law 28.3, that the addition of additive inverse behaves like a binary subtraction operator

$$
\begin{aligned}
&(x \dot{-} y) \dot{*} z \\
&\quad = (x \dot{+} \dot{-} y) \dot{*} z && \text{defn. } (\_ \dot{-} \_) \\
&\quad = x \dot{*} z \dot{+} \dot{-} y \dot{*} z && \text{distributive} \\
&\quad = x \dot{*} z \dot{+} \dot{-} (y \dot{*} z) && \text{ring minus law 28.2} \\
&\quad = x \dot{*} z \dot{-} y \dot{*} z && \text{defn. } (\_ \dot{-} \_)
\end{aligned}
$$

■

Proof of law 28.4, that if every element x in a ring obeys $x \dot{*} x = x$, then it is a Boolean ring

$$
\begin{aligned}
&x \dot{*} (x \dot{+} x) \\
&\quad = x \dot{*} x \dot{+} x \dot{*} x && \text{distributive} \\
&\quad = x \dot{+} x && \text{hypothesis}
\end{aligned}
$$

Either $x = O$ — but x is quantified over all of g, so this is not the case — or $x \dot{+} x = O$ (from a property of rings, law 28.1).

■

Proof of law 28.7, that $O \prec x \Leftrightarrow \dot{-} x \prec O$

$$
\begin{aligned}
&O \prec x && \text{assumption} \\
&\quad \Rightarrow (\dot{-} x) \dot{+} O \prec (\dot{-} x) \dot{+} x && \text{OrderedIntegralDomain axiom} \\
&\quad \Rightarrow \dot{-} x \prec O && \text{defn. } O \text{ and } \dot{-}
\end{aligned}
$$

■

Proof of law 28.8, that g is unbounded above.

Take $y' = x \dot{+} I$ ($y' \in g$ follows from closure property):

$$
\begin{aligned}
&O \prec I && \text{OrderedIntegralDomain axiom} \\
&\quad \Rightarrow x \dot{+} O \prec x \dot{+} I && \text{OrderedIntegralDomain axiom} \\
&\quad \Rightarrow x \prec x \dot{+} I && \text{defn. } O
\end{aligned}
$$

■

Proof of law 28.8, that g is unbounded below.

Take $y = x \dot{+} (\dot{-} I)$ ($y \in g$ follows from closure property):

$$
\begin{aligned}
&O \prec I && \text{OrderedIntegralDomain axiom} \\
&\quad \Rightarrow O \dot{+} (\dot{-} I) \prec I \dot{+} (\dot{-} I) && \text{OrderedIntegralDomain axiom} \\
&\quad \Rightarrow O \dot{+} (\dot{-} I) \prec O && \text{defn. } \dot{-} \\
&\quad \Rightarrow x \dot{+} O \dot{+} (\dot{-} I) \prec x \dot{+} O && \text{OrderedIntegralDomain axiom} \\
&\quad \Rightarrow x \dot{+} (\dot{-} I) \prec x && \text{defn. } O
\end{aligned}
$$

■

Proof of law 28.9, that $O \prec x \Leftrightarrow O \prec x^{\dot{-}1}$

$$
\begin{array}{rl}
O \prec x & \text{assumption} \\
\quad \Rightarrow O \prec x^{\dot{-}1} \Leftrightarrow O \mathbin{\dot{*}} x \prec x^{\dot{-}1} \mathbin{\dot{*}} x & \text{OrderedField axiom} \\
\quad \Rightarrow O \prec x^{\dot{-}1} \Leftrightarrow O \prec I & \text{defn. } \dot{*} \text{ and } \_^{\dot{-}1} \\
\quad \Rightarrow O \prec x^{\dot{-}1} & \text{OrderedField axiom}
\end{array}
$$

Similarly for $x \prec O$.

■

Proof of law 28.10, that g is dense.

Take $x = (y \mathbin{\dot{+}} y') \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1}$. We use the fact that $O \prec (I \mathbin{\dot{+}} I)^{\dot{-}1}$, which follows from law 28.9, and consider just the case that both elements are greater than O.

$$
\begin{array}{rl}
O \prec y \prec y' & \text{assumption} \\
\quad \Rightarrow y \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} \prec y' \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} & \text{OrderedIntegralDomain axiom} \\
\quad \Rightarrow y \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} \mathbin{\dot{+}} y \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} \prec y \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} \mathbin{\dot{+}} y' \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} & \\
& \text{OrderedIntegralDomain axiom} \\
\quad \Rightarrow (y \mathbin{\dot{+}} y) \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} \prec (y \mathbin{\dot{+}} y') \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} & \text{distributive} \\
\quad \Rightarrow y \mathbin{\dot{*}} (I \mathbin{\dot{+}} I) \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} \prec (y \mathbin{\dot{+}} y') \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} & \text{distributive, defn. } I \\
\quad \Rightarrow y \prec (y \mathbin{\dot{+}} y') \mathbin{\dot{*}} (I \mathbin{\dot{+}} I)^{\dot{-}1} & \text{defn. } \_^{\dot{-}1} \\
\quad \Rightarrow y \prec x & \text{defn. } x
\end{array}
$$

We can similarly show that $x \prec y'$, as required. The case where y and y' are both less than O follows in a similar manner. If $y = O$ or $y' = O$, the arguments carry through. If $y \prec O \prec y'$, then $x = O$ is a suitable choice.

■

Proof of law 29.6, that there is a bijection between the integers and natural numbers:

One such bijection is $(\ \lambda\, i : \mathbb{Z}_{\_} \bullet \text{-}2 * i + 1\ ) \cup (\ \lambda\, i : \mathbb{N} \bullet 2 * i\ )$

■

Proof of law 29.9, that there is a bijection between the natural numbers and pairs of naturals:

Law 21.44 shows that it is sufficient to show the existence of a total injection in both directions.

• Case 1: Consider $(\ \lambda\, n : \mathbb{N} \bullet (n, n)\ ) \in \mathbb{N} \rightarrowtail \mathbb{N}^{2\times}$

• Case 2: Consider $(\ \lambda\, n, m : \mathbb{N} \bullet 2 ** n * 3 ** m\ ) \in \mathbb{N}^{2\times} \rightarrowtail \mathbb{N}$

■

Proof of law 29.9, that there is a bijection between the natural numbers and the rationals:

Law 21.44 shows that it is sufficient to show the existence of a total injection in both directions.

• Case 1: Consider $\mathrm{id}\,\mathbb{N} \in \mathbb{N} \rightarrowtail \mathbb{Q}$

• Case 2: Consider

$$g = \{0 \mapsto (0,0)\} \cup \{\ q : \mathbb{Q};\ n, m : \mathbb{N} \mid \mathrm{coprime}(n,m) \wedge q = n \div m \bullet q \mapsto (n,m)\ \} \\ \in \mathbb{Q} \rightarrowtail \mathbb{N}^{2\times}$$

Compose g and a bijection between pairs and naturals, to construct the required injection $\mathbb{Q} \rightarrowtail \mathbb{N}$.

■

Proof of law 29.12, that there is a bijection between the power set of natural numbers and the reals:

Law 21.44 shows that it is sufficient to show the existence of a total injection in both directions.

• Case 1: $\exists f : \mathbb{P}\,\mathbb{N} \rightarrowtail \mathbb{R} \bullet \mathit{true}$

We use a Gödelisation to construct a distinct real from each set of naturals: the $2n$th position in the binary expansion of the real is 1 precisely when n is in the particular set of naturals; all the odd positions in the binary expansion are zero (which avoids the problem of 0.1 and 0.01111... representing the same number). So take

$f = \lambda\, a : \mathbb{P}\,\mathbb{N} \bullet / + (\ \lambda\, n : a \bullet 2{**}(\text{-}2 * n)\ )$

• Case 2: $\exists f : \mathbb{R} \rightarrowtail \mathbb{P}\,\mathbb{N} \bullet \mathit{true}$: Take $f = \{\ x : \mathbb{R} \bullet x \mapsto \{q : \mathbb{Q} \bullet x < q\ \}\ \}$

This is injective (each real maps to a different set of rationals) because the rationals are dense in $\mathbb{R}$.

■

Proof of law 30.42, that p is prime precisely when p divides $(p-1)! + 1$

When $p = 2$ we have $(1! + 1) \bmod 2 = 2 \bmod 2 = 0$. So in what follows we assume $p > 2$.

$$\begin{aligned} & p \in \mathrm{prime} \\ & \quad \Leftrightarrow (\mathrm{factorial}(p-1) + 1) \bmod p = 0 \\ & \quad \Leftrightarrow \mathrm{factorial}(p-1) \bmod p = p - 1 \\ & \quad \Leftrightarrow \mathrm{factorial}(p-2) \bmod p = 1 \end{aligned}$$

We take the two cases of the double implication in turn:

- $\mathrm{factorial}(p-2) \bmod p = 1 \Rightarrow p \in \mathrm{prime}$
  Assume $p \notin \mathrm{prime}$, so $p = n * m$. We take the two cases $n = m$ and $n \neq m$ separately.
  - $n = m$, p is a square.
    Consider $n = 2$, $p = 4$. Then $(p-2)! = 2 \neq 1$.
    If p is a square > 4, then p must divide $(p-2)!$, because the factorial includes both n and $2 * n$. Contradiction.
  - $n \neq m$, with n being the smaller factor, so $n * n \leq p$. Then p must divide $(p-2)!$, because the factorial includes both n and m. Contradiction.

So $p \in \text{prime}$.

- $p \in \text{prime} \Rightarrow \text{factorial}(p-2) \bmod p = 1$
  Multiplication modulo p forms a group (law 23.4), so there is an inverse function:

$$\forall\, n : 1 \,..\, p-1 \bullet \exists\, m : 1 \,..\, p-1 \bullet (n * m) \bmod p = 1$$

We have that $((p-1)*(p-1)) \bmod p = (p*p-2*p+1) \bmod p = 1$, and also that $(1*1) \bmod p = 1 \bmod p = 1$.

No other elements are self inverse: Consider an element $n : 2..\sqrt{p}$. $n*n \in 4..p$, so $(n*n) \bmod p \neq 1$. So all elements $1 < n < \sqrt{p}$ must have a (unique) inverse $\sqrt{p} < m < p-1$. And hence all elements $\sqrt{p} < m < p-1$ have an inverse $1 < n < \sqrt{p}$.

So we have

$$\forall\, n : 2 \,..\, p-2 \bullet \exists\, m : 2 \,..\, p-2 \mid m \neq n \bullet (n * m) \bmod p = 1$$

So all the (even number of) factors in $2 \,..\, p-2$ can be paired up with their inverses, and hence vanish.

■

Proof of law 31.5, that $\_^n$ is $\subseteq$-order preserving.

Base case, $n = 0$:
$r^0 = \text{id}\, X = s^0$
Case $n > 0$; inductive hypothesis $r^n \subseteq s^n$:

$$\begin{aligned} & r^{n+1} \\ & \quad = r \fatsemi r^n \\ & \quad \subseteq s \fatsemi s^n && \text{law 20.22 with } r \subseteq s \text{ and } r^n \subseteq s^n \\ & \quad \subseteq s^{n+1} \end{aligned}$$

■

Proof of law 31.7, that iteration preserves commutativity.

Base case $n = 0$ (and hence $m = 0$ by renaming):

$r^0 \fatsemi s$
$= \mathrm{id}\, X \fatsemi s$ — definition r^0
$= s$ — property of id
$= s \fatsemi \mathrm{id}\, X$
$= s \fatsemi r^0$

Case $n = 1;\ m > 0$; inductive hypothesis $r \fatsemi s^m = s^m \fatsemi r$:

$r \fatsemi s^{m+1}$
$= r \fatsemi s \fatsemi s^m$ — definition s^{m+1}
$= s \fatsemi r \fatsemi s^m$ — theorem hypothesis
$= s \fatsemi s^m \fatsemi r$ — induction hypothesis
$= s^{m+1} \fatsemi r$

Case $n > 1$; inductive hypothesis $r^n \fatsemi s^m = s^m \fatsemi r^n$:

$r^{n+1} \fatsemi s^m$
$= r \fatsemi r^n \fatsemi s^m$ — definition r^{n+1}
$= r \fatsemi s^m \fatsemi r^n$ — induction hypothesis
$= s^m \fatsemi r \fatsemi r^n$ — case $n = 1$
$= s^m \fatsemi r^{n+1}$

■

Proof of law 31.8, that iteration preserves distributivity.

Base case $n = 0$:

$(r \fatsemi s)^0$
$= \mathrm{id}\, X$ — definition r^0
$= \mathrm{id}\, X \fatsemi \mathrm{id}\, X$ — property of id
$= r^0 \fatsemi s^0$

Case $n > 0$; inductive hypothesis $(r \fatsemi s)^n = r^n \fatsemi s^n$:

$(r \fatsemi s)^{n+1}$
$= r \fatsemi s \fatsemi (r \fatsemi s)^n$ — definition r^{n+1}
$= r \fatsemi s \fatsemi r^n \fatsemi s^n$ — induction hypothesis
$= r \fatsemi r^n \fatsemi s \fatsemi s^n$ — previous law 31.7
$= r^{n+1} \fatsemi s^{n+1}$

■

Proof of law 36.13, that the chain of prefix lower bounds has a glb.

Prefix lower bounds form a non-empty enumerable chain (law 36.12), and hence an enumerable order. (_ prefix _) is also a reflexive order (law 36.1). Hence prefix lower bounds form a well order (law 35.6). A well order has a minimum (by definition). An order is antisymmetric (by definition). Hence that minimum is the glb (law 26.27).

■

Appendix E

Index

÷, 317
#, 326
$\vec{\#}$, 328
&, 90
′, 103
$_{-\oplus}$, 313
⊕, 184
⊕/, 387
⊖, 132
∅, 116
∗, 312
$_{-}^{*}$, 251
∗/, 231
∗∗, 344
$\dot{*}$, 297
+, 311
$_{-+}$, 313
$_{-}^{+}$, 251
+/, 230
$_{-\pm}$, 313
$\dot{+}$, 297
, , , 49
−, 317
$_{-\_}$, 313
-, 312
→, 202
↠, 204
$\dot{-}$, 297
↦, 200
⤅, 204
⇸, 204
⤀, 205
$_{-}^{-1}$, 315
¬ , 55, 77
.., 324
(. . .), 68
⦇ . . . ⦈, 85
⟪. . .⟫, 90
(⦇ ⦈), 169
{. . .}, 71
⌈. . .⌉, 321
⌊. . .⌋, 321
⟨. . .⟩, 354
$_{n}\langle . . . \rangle$, 354
⟦. . .⟧, 396
∧, 55, 77
∨, 55, 77
$_{-}^{2\times}$, 113
⌊, 379
⌉, 378
⌋, 378
⌈, 379
⊢, 48
↦, 154
⊢?, 47
::=, 90
; , 55
$\overset{\circ}{9}$, 79
$\overset{\circ}{9}$/, 388
$\overset{\leftarrow}{\overset{\circ}{9}}$, 192
$\overset{\circ}{9}$, 187
$\overset{\rightarrow}{\overset{\circ}{9}}$, 192
<, 312
◁, 161
⋬, 161
↔, 153
⇻, 153
≤, 312
⇔, 57, 77
=, 62
≈, 182
≠, 113
==, 44, 46
⇒, 57, 77

$>$, 318
$\rhd$, 166
$\trianglerighteq$, 166
$\rightarrowtail$, 207
$\twoheadrightarrow$, 207
$\mapsto$, 207
$\rightarrowtail\!\!\!\!\twoheadrightarrow$, 207
$\twoheadrightarrow\!\!\!\!\twoheadrightarrow$, 207
$\rightarrowtail\!\!\!\!\twoheadrightarrow\!\!\!\!\twoheadrightarrow$, 207
$\geq$, 318
$\gg$, 79
?, 103
!, 103
$\mathbb{A}$, 311
$\forall$, 58, 75
$\bot$, 111
$\exists$, 59, 75
$\exists_1$, 61, 75
$\mathbb{F}$, 145
$\mathbb{F}_1$, 145
$\mathbb{N}$, 311
$\mathbb{N}_1$, 313
$\mathbb{N}_2$, 331
Π, 341
$\mathbb{P}$, 94
$\mathbb{P}_1$, 141
$\mathbb{Q}$, 315
$\mathbb{R}$, 316
Σ, 339
$\top$, 111
$\mathbb{Z}$, 313
_, 49
$\frown$, 355
$\frown/$, 389
$\setminus$, 80
$\setminus$, 126
$\surd$, 333
$\surd\!\!\!-$, 333
$\subset$, 140
$\subseteq$, 137
χ, 64
$\in$, 62
$\notin$, 112
λ, 66
μ, 67
$\_^n$, 334
$\cap$, 122
$\bigcap$, 122
$\circ$, 187
$\circ/$, 388
π, 347
σ, 110
θ, 83
$\cup$, 117
$\uplus$, 394
$\bigcup$, 117
$\times$, 96
$\sim$, 154
$\_^{2\times}$, 113

$\mathbb{A}$, 311
$\forall$, 58, 75
Abel, Niels Henrik, 222
AbelianGroup, 226
AbelianMonoid, 223
AbelianSemiGroup, 222
abs, 320
acyclic, 258
&, 90
anti-chain, 272
antisymmetric, 247
arithmos, 311
axiom of ancestry, 410
`AxiomaticDeclaration`, 43, 44

$\mathbb{B}$, 111
bag, 392
bag display, 396
bag sum, 394
bagcompose, 393
bicompose, 196
bijection, 207
`BindingConstruction`, 74, 83
`BindingExtension`, 74, 85
`BindingSelection`, 74, 88
Boole, George, 111
BooleanRing, 298
bound, greatest lower, 283
bound, least upper, 283
bound, lower, 172
bound, upper, 172

calendar, 86
cardinality, 326
cardinality, total, 328

Cartesian diagram, 449
Cartesian square, 113
`CartesianProduct`, 65, 96
ceiling, 321
chain, 276
chain_1, 276
characteristic tuple, 64
χ, 64
closure property, 135
closure, reflexive transitive, 251
closure, symmetric, 247
closure, transitive, 251
coextract, 378
cofilter, 379
common logarithm, 346
comparable, 270
compatible relations, 182
complete distributed product, 341
complete distributed sum, 339
CompleteField, 304
component, 264
composite, 127, 331
composition, distributed functional, 388
composition, distributed relational, 388
composition, functional, 187
composition, left demonic, 192
composition, relational, 187
composition, right demonic, 192
composition, schema, 79
concatenation, distributed, 389
concetenation, 355
`ConditionalExpression`, 65, 68
`Conjecture`, 43, 47
conjunction, 55
conjunction, low-precedence, 55, 104
conjunction, schema, 77
connected, 264
connected_1, 264
coprime, 331
cos, 347
cosine, 347
count, 392

de Morgan's laws, 56, 59, 77, 175, 178
de Morgan, Augustus, 56
decoration stroke, 103
degree, 336
demonic composition, 192
Descartes, René, 449
Dijkstra, Edsgar, 260
disjoint, 218
disjunction, 55
disjunction, schema, 77
display, bag, 396
display, sequence, 354
display, stream, 354
distributed concatenation, 389
distributed functional composition, 388
distributed override, 387
distributed product, complete, 341
distributed product, finite, 231
distributed relational composition, 388
distributed sum, complete, 339
distributed sum, finite, 230
distributeOverLabelledSet, 228
distributeOverSeq, 386
distribution properties, 133
distribution property, 142
div, 322
division, integer, 322
do, 261
dom, 158
domain, 158
domain restriction, 161
domain subtraction, 161
dual laws, 156

$\exists$, 59, 75
$\exists_1$, 61, 75
empty set, 116
enumerableChain, 359
enumerableOrder, 358
enumerate, 360
equality, 62
`EqualityDeclaration`, 44
equivalence, 57
equivalence class, 258
equivalence relation, 257
equivalence, schema, 77
Euclid, 261
even, 127
existential quantification, 59
existential schema quantification, 75
exp, 344
exponential function, 344
`Expression`, 44, 46, 54, 65

exterior vertex, 236
extract, 378
extremum, 236

$\mathbb{F}$, 145
$\mathbb{F}_1$, 145
factor, 399
factorial, 342
false, 55
fbag, 392
Field, 301
filter, 379
finite distributed product, 231
finite distributed sum, 230
finite function, 204
finite injection, 207
finite relation, 153
finite subset, 145
finite surjection, 205
finite surjective injection, 207
finiteness predicate, 145
first component, 152
floor, 321
forest, 266
formSequence, 360
`FreeType`, 43, 90
front, 375
function, 200
function keyword, 49
function, finite, 204
function, partial, 200
function, total, 202
functional composition, 187
functional composition, distributed, 388
functionalAt, 199
Fundamental Theorem of Arithmetic, 332

Galois, Evariste, 304
gcd, 400
generalised set intersection, 122
generalised set union, 117
generic keyword, 49
`GenericAxiomaticDeclaration`, 43, 46
`GenericConjecture`, 43, 47
`GenericHorizontalDeclaration`, 43, 46
`GenericOperatorDeclaration`, 43
`GenericSchemaBoxDeclaration`, 43, 46
`GivenSet`, 43, 89
glb, 283
GraphPreservingInjection, 287
GraphPreservingMap, 287
GraphReversingInjection, 290
GraphReversingMap, 290
greatest common divisor, 400
greatest lower bound, 283
Gregory XIII, 86
Group, 226

hard newline, 104
head, 375
hiding, schema, 80
homomorphism, 212
`HorizontalDeclaration`, 43, 44

id, 239
idempotent, 221, 250, 252
identity relation, 239
if-then-else expression, 68
image, lower, 168
image, lower singleton, 181
image, relational, 169
image, upper, 168
image, upper singleton, 180
implication, 57
implication, schema, 77
incomparable, 270
inDegree, 336
induction principle, 145, 286
induction, transfinite, 286
inequality, 113
infix, 365
injection, 207
injection, finite, 207
injection, finite surjective, 207
injection, partial, 207
injection, partial surjective, 207
injection, total, 207
integer division, 322
integer part, 321
integer range, 324
IntegerProperties, 312
IntegralDomain, 300
interior vertex, 236
intersection, 122
intersection, generalised, 122
intransitive, 254

intransitive residue, 255
inverse, relational, 154
ipath, 381
irreflexive, 243
irreflexiveEnumerableOrder, 358
irreflexiveOrder, 271
irreflexiveTotalOrder, 274
iseq, 353
isequence, 351
isomorphism, 214
istr, 352
istream, 349
items, 395
iteration, maximal, 261
iteration, relation, 334

Julius Caesar, 86

λ, 66
`LambdaExpression`, 65, 66
last, 375
lcm, 400
leaf, 236
leap year, 86
least common multiple, 400
least upper bound, 283
left demonic composition, 192
leftassoc keyword, 49
`LetExpression`, 65, 67
line breaking, 104
ln, 345
locallyFinite, 256
locallyFiniteIn, 256
locallyFiniteOut, 256
$\log_n$, 346
logarithm, common, 346
logarithm, natural, 345
low-precedence conjunction, 55, 104
lower bound, 172
lower bound, greatest, 283
lower image, 168
lower shadow, 177
lower singleton image, 181
lub, 283

μ, 67
makeComplete, 338
maplet, 154
max, 330
maximal iteration, 261
maximum, 236, 281
mean, 397
median, 398
membership, 62
merge, 195
min, 330
minimum, 236, 281
mitochondrial Eve, 410
mod, 322
mode, 399
modulus, 322
Monoid, 223
`MuExpression`, 65, 67

$\mathbb{N}$, 311
$\mathbb{N}_1$, 313
$\mathbb{N}_2$, 331
natural logarithm, 345
negation, 55
negation, schema, 77
network diagram, 452
newline, hard, 104
newline, soft, 104
node, 236
non-empty finite subset, 145
non-empty subset, 141
non-membership, 112
normalised schema, 110

odd, 127
`OperatorTemplate`, 43, 49
order, 270
OrderedField, 303
OrderedIntegralDomain, 302
outDegree, 336
override, 184
override, distributed, 387

Π, 341
$\mathbb{P}$, 94
π, 347
$\mathbb{P}_1$, 141
`Paragraph`, 42, 43
partial, 269
partial function, 200
partial injection, 207

partial surjection, 204
partial surjective injection, 207
partition, 219
path, 381
pi, 347
pigeonhole principle, 146
piping, schema, 79
Pope Gregory XIII, 86
poset, 273
poset_1, 273
power function, completed, 345
power function, integer exponent, 343
`PowerSet`, 65, 94
pre keyword, 80
precondition, schema, 80
`Predicate`, 47, 54
predicate paragraph, 14
predicate, schema, 62
prefix, 364
preorder, 277
prime, 127, 331
product, complete distributed, 341
product, finite distributed, 231
productbag, 397
projection, schema, 80
proper subset, 140

$\mathbb{Q}$, 315
quantification, existential, 59
quantification, existential schema, 75
quantification, unique existential, 61
quantification, unique existential schema, 75
quantification, universal, 58
quantification, universal schema, 75

$\mathbb{R}$, 316
ran, 158
range, 158
range restriction, 166
range subtraction, 166
range, integer, 324
RationalProperties, 315
RealProperties, 316
reflexive, 241
reflexive transitive closure, 251
reflexiveEnumerableOrder, 358
reflexiveOrder, 271
reflexiveTotalOrder, 274
`Relation`, 54, 62
relation arrow, 153
relation iteration, 334
relation keyword, 49
relation, finite, 153
relation, identity, 239
relational composition, 187
relational composition, distributed, 388
relational image, 169
relational inverse, 154
relational override, 184
relations, compatible, 182
remainder, 322
residue, intransitive, 255
restriction, domain, 161
restriction, range, 166
rev, 374
reverse, 374
right demonic composition, 192
rightassoc keyword, 49
Ring, 298
root, 236, 238
Russell, Bertrand, 234

Σ, 339
σ, 110
schema composition, 79
schema conjunction, 77
schema disjunction, 77
schema equivalence, 77
schema hiding, 80
schema implication, 77
schema negation, 77
schema piping, 79
schema precondition, 80
schema predicate, 62
schema projection, 80
`SchemaBoxDeclaration`, 43, 44
`SchemaCombination`, 74, 79
`SchemaConstruction`, 74, 83
`SchemaExpression`, 65, 74
`SchemaPropositional`, 74, 77
`SchemaQuantification`, 74, 75
`SchemaRenamingExpression`, 74, 82
`SchemaRestriction`, 74, 80
`SchemaText`, 44, 46, 54, 64
second component, 152
`Section`, 42

SemiGroup, 222
seq, 353
sequence, 350
sequence display, 354
set difference, 126
set difference, symmetric, 132
`set display`, 71
set intersection, 122
set intersection, generalised, 122
set union, 117
set union, generalised, 117
`SetComprehension`, 65
`SetExtension`, 65, 71
shadow, lower, 177
shadow, upper, 176
shift, 354
sign, 319
signature, 110
sin, 347
sine, 347
singleton image, lower, 181
singleton image, upper, 180
sink, 236
sizebag, 397
soft newline, 104
Sort, 295
source, 236
`Specification`, 42
split, 195
square root, 333
squash, 361
StableSort, 296
steps, 382
str, 352
stream, 349
stream display, 354
stroke, 103
stronglyConnected, 264
subscript stroke, 103
subset, 137
subset, finite, 145
subset, non-empty, 141
subset, non-empty finite, 145
subset, proper, 140
subtraction, domain, 161
subtraction, range, 166
suffix, 364
sum, bag, 394
sum, complete distributed, 339
sum, finite distributed, 230
sumbag, 397
surjection, 204
surjection, finite, 205
surjection, partial, 204
surjection, total, 204
surjective injection, 207
symmetric, 245
symmetric closure, 247
symmetric set difference, 132

tail, 375
θ, 83
Thebes, 411
time machine, 429
total cardinality, 328
total function, 202
total injection, 207
total surjection, 204
totalOrder, 274
transfinite induction', 286
transitive, 249
transitive closure, 251
tree, 267
true, 55
`TupleComponentSelection`, 65, 69
`TupleExpression`, 65, 68

union, 117
union, generalised, 117
unique existential quantification, 61
unique existential schema quantification, 75
universal quantification, 58
universal schema quantification, 75
upper bound, 172
upper bound, least, 283
upper image, 168
upper shadow, 176
upper singleton image, 180

Venn diagram, 447
Venn, John, 447
vertex, 236, 237
vertex degree, 336

wellFoundedChain, 286

wellFoundedChain$_1$, 286
wellOrder, 285
wellRooted, 238

$\times$, 96

$\mathbb{Z}$, 313