A TUTORIAL ON PROOF IN STANDARD Z

by

Stephen Brien and Andrew Martin

Technical Monograph PRG-120
ISBN 0-902928-94-5

February 1996

Oxford University Computing Laboratory
Programming Research Group
Wolfson Building, Parks Road
Oxford OX1 3QD
England

Oxford University Computing Laboratory
Programming Research Group
Wolfson Building, Parks Road
Oxford OX1 3QD
England

Z User Meeting '95, Limerick

A Tutorial on Proof in Standard Z

Stephen Brien and Andrew Martin

5th September 1995

Introduction

In these notes we present material designed to support an explanation of how to conduct proofs using the deductive system presented in the draft Z Standard. This is, of course one of several possible deductive systems for Z. We aim to present an account of the various components of the deductive system, showing how they are used together, and how they permit the formal proof of theorems involving sizeable Z specifications.

The method of proof is supported by Jigsaw, a theorem proving assistant into which the deductive system of standard Z has been incorporated. We further aim to show how Jigsaw's support of the tactic language *Angel* allows proofs to be defined in a more general and reusable way.

An appendix gives the relevant sections of the current draft standard We do not aim to explain every rule, but merely enough to allow the reader to read, understand, and use the standard's deductive system. Our work on logics for Z has been greatly helped by collaboration with Jim Woodcock, and review from other members of the Z Standards Panel. In particular we have benefited from the work of Jones (1990) and Harwood (1990).

Most of the material used here is derived from the project *Models, Algebra, and Mechancial Support in Z*, funded by EPSRC grant number GR/J46630.

Contents

Part I
Motivation, and Simple Proofs

This first part of the tutorial illustrates the issues arising from conducting proofs by means of Spivey's well-known *BirthdayBook* example. We then introduce the deductive system used in the standard (a Gentzen-style sequent calculus) and show how to construct proofs using it. The familiar rules of the propositional calculus are given to illustrate the format of rules.

Having provided a basic minimum of deductive rules, we can develop tactics for completing proofs. We show how tactics in *Angel* are constructed and how to build a general tactics to simplify propositions and solve tautologes.

Contents

1 Specifications and Proofs

Most readers will be familiar with Spivey's *BirthdayBook* specification (Spivey 1992), and so we use it as a running example here. Recall that two sets are given, denoting the collection of people whose birthdays are to be recorded, and the set of possible birthdays:

$$[NAME, DATE]$$

The state of the system is given by a schema *BirthdayBook*, which has two components: a mapping of names to dates, and a set of those people whose birthdays are known. These components are linked via an invariant.

BirthdayBook

$$known : \mathbb{P}\, NAME;$$
$$birthday : NAME \pfun DATE$$
$$known = \mathrm{dom}\, birthday$$

Initially, no birthdays are known.

InitBirthdayBook

$$BirthdayBook$$
$$known = \varnothing$$

We provide an operation to add a birthday to the book. The operation takes as input a person's name and birthday, and succeeds providing the name is not already known, updating the state, but producing no output.

AddBirthday

$$\Delta BirthdayBook;$$
$$name? : NAME;$$
$$date? : DATE$$
$$\neg\, (name? \in known) \land$$
$$birthday' = birthday \cup \{(name?, date?)\}$$

1.1 Theorems of the Specification

We define an entailment relation between a specification and a predicate, to mean that the specification guarantees the truth of the predicate.

A *Sequent* comprises a specification, followed by an assertion sign (the 'turnstile'), followed by a predicate:

$$Sequent \quad ::= \quad Spec \vdash Pred \quad .$$

Thus a sequent appears as:

$$\Pi_1 \ddagger \cdots \ddagger \Pi_n \vdash P$$

These paragraphs provide a context for the predicate on the right-hand side, (for the sequent to be well-formed, the free variables of P must be declared in $\Pi_1 \ddagger \cdots \ddagger \Pi_n$) and may be considered to be (part of) the specification in which the conjecture is evaluated.

Any of the paragraphs of the specification can be explicitly included in the antecedent as follows:

$$(\Gamma \ddagger \Pi)\, \Gamma_1 \vdash P \quad \equiv \quad (\Gamma)\, \Pi \ddagger \Gamma_1 \vdash P \quad .$$

We can now use this sequent notation to express theorems about the birthday book. The initial state may be shown to satisfy the state invariant:

$$\vdash \exists \mathit{InitBirthdayBook} \bullet \mathit{true}$$

The schema *AddBirthday* entails an update of *known* as well as of *birthday*.

$$\mathit{AddBirthday} \vdash \mathit{known}' = \mathit{known} \cup \{\mathit{name}?\}$$

1.2 Informal Proofs

Spivey provides an informal proof of the theorem about *AddBirthday* using an equational reasoning proof style as follows:

$$\begin{array}{lr}
\mathit{known}' = \mathrm{dom}\, \mathit{birthday}' & \text{[invariant after]} \\
\quad = \mathrm{dom}(\mathit{birthday} \cup \{\mathit{name}? \mapsto \mathit{date}?\}) & \text{[spec of } \mathit{AddBirthday}\text{]} \\
\quad = \mathrm{dom}\, \mathit{birthday} \cup \mathrm{dom}\{\mathit{name}? \mapsto \mathit{date}?\} & \text{[fact about 'dom']} \\
\quad = \mathrm{dom}\, \mathit{birthday} \cup \{\mathit{name}?\} & \text{[fact about 'dom']} \\
\quad = \mathit{known} \cup \{\mathit{name}?\} & \text{[invariant before]}
\end{array}$$

This proof cogently displays the top level reasoning required to justify the theorem. Its clarity and simplicity is based on the fact that it can call on obvious facts about the specification and on general properties of the *dom* operator. Such appeals to 'obvious' facts are entirely appropriate when considering such a small and well understood example.

1.3 Why Formalize?

A proof of a property of a larger and more complex specification will not be able to appeal so easily to obvious facts. The larger a proof becomes the more the need for formality to provide the necessary assurance of correctness. However, the problem with adopting a fully formal approach to proof is that there is a large amount of exacting checking that needs to be done. This checking comprises ensuring that rules are properly applied, the discharging of provisos, and trivial type checking. Such an overhead quite naturally creates quite a significant disincentive for anyone to tackle a substantial proof in a formal manner.

The support that can be provided by theorem proving assistants such as Jigsaw and Isabelle makes the task of formal proof much simpler and also increases assurance of correctness. In this tutorial we shall present the logic for Z that is in the draft standard and show how it is supported in Jigsaw. The support for proof provided in Jigsaw comes in three forms. Firstly, the simple mechanical application of the rules, secondly the automatic discharging of provisos, and thirdly the provision of a tactic language for describing and directing proofs.

2 Simple Rules And Proofs

The deductive system we use is a Gentzen-style sequent calculus based on $\mathcal{W}$, a logic for Z (Woodcock and Brien 1992), though we use only one consequent. The deductive system consists of a number of rules for manipulating sequents, and a method of combining rules to generate proofs.

2.1 Rules

Inference rules will be written as follows:

$$\mathit{Rule} ::= \frac{\mathit{Premises}}{\mathit{Conclusion}}\ \mathit{Name}$$

The premises are a (possibly empty) list of sequents:

$$\mathit{Premises} ::= \mathit{Sequent} \ldots \mathit{Sequent}$$

The conclusion is always a single sequent:

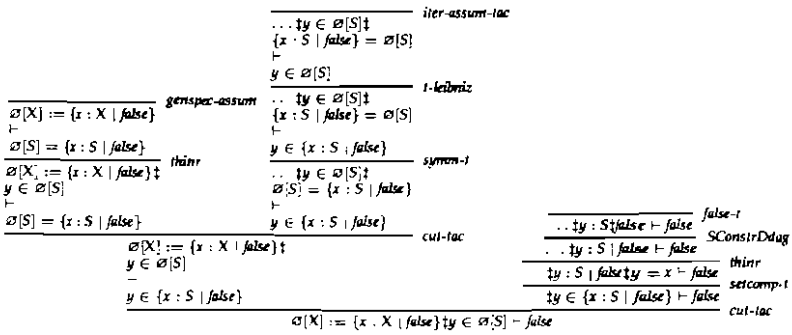
$$\mathit{Conclusion} ::= \mathit{Sequent}$$

2.2 Proofs

Proofs in the deductive system proceed in the way that is usual for sequent calculi: proofs are developed *backwards*, starting from the sequent which is to be proved. A rule is applied, resulting in fresh sequents which must be proved. This process continues until there are no more sequents requiring proof, in which case the original sequent is now proved.

Proof Trees A completed proof may thus be represented as a tree, with the proved sequent as the root node, and every leaf node containing an empty list of sequents.[1] An example of such a tree follows; its contents will be explained later.

[1] However, if some of these lists in the leaves are non-empty, then the derivation tree is still useful, although it does not represent a proof, it represents a partial proof.



Theorems A sequent is a theorem if there is a completed proof for it. The relationship between theorems and completed proofs is given as follows:

The rule $\frac{}{\Gamma \vdash P}$ is sound if and only if the sequent $\Gamma \vdash P$ is a theorem.

2.3 Structural Rules

The first rules that we consider are those that allow us to manipulate the information contained in the antecedent. We shall consider the case where the only types of paragraphs in the antecedent are constraint paragraphs. They are in effect simple predicates.

Assumption The most general way of completing a proof is to arrive at a stage where the predicate to be proved is contained in the definition of the specification that is being assumed.

$$\frac{}{\Gamma \ddagger P \vdash P}\ \mathit{AssumPred}$$

Thinning Any theorem that can be proved using some set of axioms can also be proved with extra ones added. The first thinning rule states that it is always safe to thin from the left of a specification in the antecedent. This represents the fact that the information is built up incrementally. The second thinning rule states that a predicate can always be safely removed from the right of a specification.[2]

$$\frac{\Gamma \vdash P}{\Pi \ddagger \Gamma \vdash P}\ \mathit{Thinl} \qquad \frac{\Gamma \vdash P}{\Gamma \ddagger Q \vdash P}\ \mathit{Thinr}$$

Shifting The order of the predicates in the antecedent can be shifted

$$\frac{\Gamma \ddagger P_q \ddagger P_1 \ddagger \cdots \ddagger P_n \vdash R}{\Gamma \ddagger P_1 \ddagger \cdots \ddagger P_n \ddagger Q \vdash R}\ \mathit{Shift}$$

[2]There are extra conditions required for the general paragraph case.

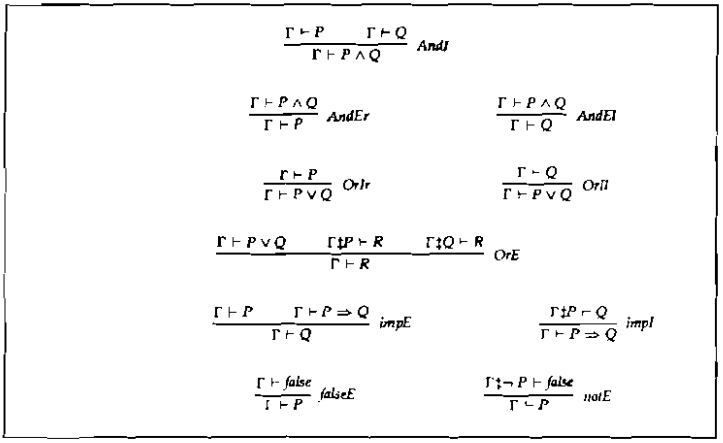


Figure 1: The Propositional Calculus

2.4 Propositions

The definition of the boolean algebra using deduction rules is presented in figure 1.

Proofs of Propositions Using the rules of the propositional calculus, we can construct proofs for propositional tautologies. For example, the commutativity of $\wedge$:

$$\dfrac{\dfrac{\dfrac{\overline{P \ddagger Q \vdash Q}\ \textit{assum-tac}}{P \wedge Q \vdash Q}\ \textit{and-l} \quad \dfrac{\dfrac{\overline{P \vdash P}\ \textit{assum-tac}}{P \ddagger Q \vdash P}\ \textit{thinr}}{P \wedge Q \vdash P}\ \textit{and-r}}{P \wedge Q \vdash Q \wedge P}\ \textit{AndI}}{\vdash P \wedge Q \Rightarrow Q \wedge P}\ \textit{impI}$$

2.5 Derived Rules

We can generate derived rules from partial proof trees. Once derived, and given a name, a rule can be reused, in different circumstances. The construction of a derived rule is based on the notion of tree-squashing.

Tree Squashing Suppose that we have the derivation tree:

$$\dfrac{S_1 \quad \ldots \quad \dfrac{S_{i1} \quad \ldots \quad S_{im}}{S_i}\ [R_i] \quad \ldots \quad S_n}{Seq}\ [R]$$

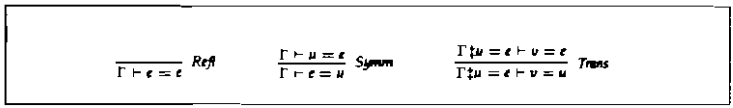


Figure 2: Equality

where each of the rules R and R_i are sound rules, then the derived rule

$$\frac{S_1 \quad \ldots \quad S_{i1} \quad \ldots \quad S_{im} \quad \ldots \quad S_n}{Seq}\ [R']$$

is also sound. By repeating this application many times a large tree can be compressed into a compact rule. A simple example of a derived rule is the cut rule.

Cut Rule The *cut rule* is used to structure proofs into lemmas: it permits the addition of hypotheses to the antecedent; these hypotheses may be discharged separately:

$$\frac{\Gamma \ddagger P \vdash Q \quad \Gamma \vdash P}{\Gamma \vdash Q}\ Cut$$

This rule is derived from the proof-tree that combines implication introduction and elimination:

$$\frac{\dfrac{\Gamma \ddagger P \vdash Q}{\Gamma \vdash P \Rightarrow Q}\ impI \qquad \Gamma \vdash P}{\Gamma \vdash Q}\ impE$$

It is the responsibility of the user of the cut rule (and those for implication and disjunction elimination) to ensure that the well-typedness of the sequent is preserved by the addition of new predicates. New declarations can be cut in using an existentially quantified predicate.

2.6 Equality

In order to provide a basic language with which to reason, we assume that there are expressions with an equality relation between them. Equality is a reflexive, commutative and transitive relation. These rules are given in figure 2.

Proofs using Equality The properties of equality can be used to prove that for the birthaday book in an intialised state, the domain of the *birthday* function is empty:

$$InitBirthdayBook \vdash \mathrm{dom}\, birthday = \varnothing$$

Having extracted the property of the schema we can complete the proof as follows:[3]

[3]The rules for extracting the property of a schema will be given later.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdots \dagger known = \mathrm{dom}\, birthday \vdash known = \mathrm{dom}\, birthday}\,Refl}{\cdots \dagger known = \mathrm{dom}\, birthday \vdash \mathrm{dom}\, birthday = known}\,Symm}{\cdots \dagger known = \mathrm{dom}\, birthday \dagger \varnothing = known \vdash \mathrm{dom}\, birthday = known}\,thinr}{\cdots \dagger known = \mathrm{dom}\, birthday \dagger \varnothing = known \vdash \mathrm{dom}\, birthday = \varnothing}\,Trans}{\cdots \dagger known = \mathrm{dom}\, birthday \dagger known = \varnothing \vdash \mathrm{dom}\, birthday = \varnothing}\,Symm$$

3 Simple Tactics

The languages used in theorem-proving assistants to direct proofs are often called *tactics*.

In order to collect inference rules together into derived rules (proof procedures), we employ a simple tactic language which is a subset of the language *Angel*, described by Martin, Gardiner and Woodcock [1996]. The chief tactic constructions we use at this stage are *sequential composition*, and *parallel composition*. Such tactic combinators are sometimes called *tacticals*.

Tactics will be defined by paragraphs with the form

tacname := *definition*

and may be parametrized.

In this tutorial, we will frequently give proof trees as well as tactics when derived rules are discussed. Tactics may, however, describe proof procedures which are too complex (or simply too large) to be presented as trees.

3.1 Primitive Inference Rules

Use of primitive inference rules may be indicated by the keyword **rule**. In the account which follows, this will frequently be omitted, for ease of reading. Instead, we adopt a convention that inference rules are written with an initial capital letter, whereas other tactics will be entirely in lower case.

Examples of primitive inference rules already encountered are *AssumPred*, *AndI*, and *Thinr*.

3.2 Sequential Composition

Sequential composition simply entails applying inference rules (or tactics) one after the other:

simplify-iff := *IffDef* ; *AndI*

Corresponding to the proof tree

$$\dfrac{\dfrac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash Q \Rightarrow P}{\Gamma \vdash P \Rightarrow Q \land Q \Rightarrow P}\,AndI}{\Gamma \vdash P \Leftrightarrow Q}\,IffDef$$

3.3 Parallel Composition

When a proof tree *bifurcates*, parallel composition allows the application of different tactics to different portions of the tree. For example, the derived rule of *Cut* above may be represented as a tactic:

$$\textit{cut-tacp} := \textit{impEp};\ (\textit{impI} \parallel \textbf{skip})$$

After application of *impE*, the right-hand branch is left alone (**skip** is a tactic which leaves its goal unchanged), and the left-hand branch has *impI* applied to it.

$$\dfrac{\dfrac{\Gamma \ddagger P \vdash Q}{\Gamma \vdash P \Rightarrow Q}\ \textit{impI} \qquad \Gamma \vdash P}{\Gamma \vdash Q}\ \textit{impE}$$

Note that this parallel combinator is merely a *binary* operator; in the event of multiply branching trees, the tactic structure reflects the structure of the tree. For example, the rules and tactics already given can be used to simplify a conjunction *in the antecedent*:

$$\begin{array}{l}\textit{and-right} := \textit{cut-tac}(P):\\ \qquad\qquad (\textit{AndEr}(Q):\ \textit{AssumPred}\\ \qquad\qquad \parallel\\ \qquad\qquad \textit{lshift}\,2;\ \textit{cut-tac}(Q);\ (\textit{AndEl}(P);\ \textit{AssumPred} \parallel \textit{lshift}\,2;\ \textit{thinr}\,1))\end{array}$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma \ddagger P \ddagger P \wedge Q \vdash P \wedge Q}\ \begin{array}{l}\textit{AssumPred}\\ \textit{AndEr}(Q)\end{array}}{\Gamma \ddagger P \wedge Q \vdash P}\qquad \dfrac{\dfrac{\dfrac{}{\Gamma \ddagger P \ddagger P \wedge Q \vdash P \wedge Q}\ \begin{array}{l}\textit{AssumPred}\\ \textit{AndEl}(P)\end{array}}{\Gamma \ddagger P \ddagger P \wedge Q \vdash Q}\qquad \dfrac{\dfrac{}{\Gamma \ddagger P \ddagger Q \ddagger P \wedge Q \vdash R}\ \textit{thinr}\,1}{\Gamma \ddagger P \ddagger P \wedge Q \ddagger Q \vdash R}\ \textit{lshift}\,2}{\Gamma \ddagger P \ddagger P \wedge Q \vdash R}\ \textit{cut-tac}(Q)}{\Gamma \ddagger P \wedge Q \vdash R}\ \begin{array}{l}\textit{lshift}\,2\\ \textit{cut-tac}(P)\end{array}$$

However, good style will frequently mean that tactics are not presented in such a tree-structured way. Many inferences return a pair of goals, one of which is a minor condition which is easily dispatched, the other which represents, in some sense, the ongoing 'real' proof.

For example, in the above proof, after P has been provided by the cut, it might be seen as desirable to discharge the goal $P \wedge Q \vdash P$, before proceeding with the rest of the proof. Hence, an alternative, more linear structure for the tactic is as follows:[4]

$$\begin{array}{l}\textit{and-right} := \textit{cut-tac}(P);\\ \qquad\qquad (\textit{AndEr}(Q);\ \textit{AssumPred} \parallel \textbf{skip});\\ \qquad\qquad \textit{lshift}\,2;\ \textit{cut-tac}(Q);\\ \qquad\qquad (\textit{AndEl}(P);\ \textit{AssumPred} \parallel \textbf{skip});\\ \qquad\qquad \textit{lshift}\,2;\ \textit{thinr}\,1\end{array}$$

In this way, many tree-like proofs can be reduced to an essentially linear form (similar to equational reasoning)—see *eqtac* in Section 6.3, below.

[4] The equivalence of these two definitions can be proved using the tactic calculus of Martin et al. (1996).

3.4 Combining sequential and parallel

Occasionally it is useful to apply the *same* tactic to *each* branch of a bifurcating proof tree. A decorated form of sequential composition ';' is used to accomplish this. For example, following *simplify-iff*, above, each branch can be further simplified using *impI*.

$$\textit{simplify-iff}_2 := \textit{IffDef}\,;\ \textit{AndI}\ ;\ \textit{impI}$$

Corresponding to the proof tree

$$\dfrac{\dfrac{\Gamma \ddagger P \vdash Q}{\Gamma \vdash P \Rightarrow Q}\ \textit{impI} \qquad \dfrac{\Gamma \ddagger Q \vdash P}{\Gamma \vdash Q \Rightarrow P}\ \textit{impI}}{\dfrac{\Gamma \vdash P \Rightarrow Q \wedge Q \Rightarrow P}{\Gamma \vdash P \Leftrightarrow Q}\ \textit{IffDef}}\ \textit{AndI}$$

3.5 Alternation

The tactics we have seen so far are entirely deterministic; each one performs exactly one task. In order to write more general proof procedures, additional control structures are needed.

The tactic combinator '|' combines tactics in *alternation*, so that the second one is attempted only if the first fails to apply. For example the following tactic applies any introduction rule which will succeed:

$$\textit{prop-left} := \textit{AndI} \mid \textit{OrIr} \mid \textit{OrIl} \mid \textit{impI}$$

so it is capable of both the following inferences:

$$\dfrac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}\ \textit{prop-left} \qquad\qquad \dfrac{\Gamma \ddagger P \vdash Q}{\Gamma \vdash P \Rightarrow Q}\ \textit{prop-left}$$

Alternation is intended to be interpreted in an angelically nondeterministic manner.[5] That is, the choice of which rule to apply is not merely governed by which is presently the most useful, but which will be useful later in the proof (this will, in general, be accomplished via backtracking). Hence, when *prop-left*, above, is presented with a disjunction, it may choose to eliminate the left or the right disjunct—the choice of which will depend on which disjunct is needed for the remainder of the proof. This behaviour is characterised by the following tactic equivalence:

$$(t_1 \mid t_2)\ ;\ t_3 = t_1\ ;\ t_3 \mid t_2\ ;\ t_3$$

Such backtracking will sometimes be undesirable (for reasons of efficiency, or for guaranteeing termination of recursive tactics) so an operator 'cut'—written '!', and not to be confused with the logical cut rule—is provided. Backtracking is restricted to the scope of the cut.[6]

[5] In contrast to, say, the ELSE tactical of Edinburgh LCF.

[6] Hence, for non-backtracking t_1 and t_2, $!(t_1\ ;\ t_2)$ behaves like LCF's t_1 ELSE t_2.

3.6 Pattern Matching

When parametrised tactics are used, it is frequently useful to have some of the parameters extracted from the current goal—this both saves typing and adds to the generality of the tactic. For example, the tactics for *and-right* above must know the values of P and Q found in the goal to which they are applied, so that they can be cut in at appropriate points. The tactical π is used to accomplish this. The tactic $\pi e, p \bullet G \rightarrow t(e,p)$ matches the terms e and p against the current goal, according to the pattern given by G, and then behaves like tactic t, parametrised by e and p. Thus, *and-right* can be made fully general by writing

$$
\begin{aligned}
\mathit{and\text{-}t} := \pi\, \Gamma, p, q, r \bullet (\Gamma \natural p \wedge q \vdash r) \rightarrow\ & \\
& \mathit{cut\text{-}tac}(p); \\
& (\mathit{AndEr}(q);\ \mathit{AssumPred} \parallel \mathbf{skip}); \\
& \mathit{lshift}2;\ \mathit{cut\text{-}tac}(q); \\
& (\mathit{AndEl}(p);\ \mathit{AssumPred} \parallel \mathbf{skip}); \\
& \mathit{lshift}2;\ \mathit{thinr}1
\end{aligned}
$$

3.7 Common Derived Tacticals

Using the tacticals above, some common patterns of tactic application can be defined as derived tacticals.

try t applies t if possible, but succeeds whether t applies or not.

$$\mathit{try}\, t := !(t \mid \mathbf{skip})$$

exhaust t applies t as many times as is possible.

$$\mathit{exhaust}\, t := t\ ;\ \mathit{exhaust}\, t \mid \mathbf{skip}$$

exhausts generalizes *exhaust*, by applying it to *all* of the resulting goals after t is applied.

$$\mathit{exhausts}\, t := t\ ;\ \mathit{exhausts}\, t \mid \mathbf{skip}$$

4 A Tactic for Proving Tautologies

By combining the proof rules given so far and the tacticals described above, we can give a tactic which simplifies terms of the propositional calculus—and discharges goals which are tautologies.

Application of the propositional rules will be governed by a tactic *props*. This is composed using a number of tactics of the form t-$*$ and $*$-t. The former simplify terms in the consequent; the latter, those in (at the right-hand end of) the antecedent. The code for *and-t* has already been given; the others are similar.

An exception is the tactic *t-or*, which, rather than applying the *OrI* rules, implements a more obviously constructive rule:

$$\frac{\Gamma \ddagger \neg Q \vdash P}{\Gamma \vdash P \vee Q}\ t\text{-}or$$

It is also unclear what rules to use for *not-t* and *imp-t*; we choose the following:

$$\frac{\Gamma \vdash P \vee Q}{\Gamma \ddagger \neg P \vdash Q}\ not\text{-}t \qquad \frac{\Gamma \vdash R \vee P \quad \Gamma \ddagger Q \vdash R}{\Gamma \ddagger P \Rightarrow Q \vdash R}\ imp\text{-}t$$

There is a danger that *not-t* and *t-or* will create cycles, if applied indiscriminately. Therefore, *props* arranges that *not-t* will be applied only if all of the *t-** have failed (so Q is 'atomic'—its outermost connective is not one of the five propositional connectives), *and* after application of *not-t*: *t-or*, one of the *t-** succeeds—thereby ensuring that P is *not* atomic.

$$\begin{aligned} props := {} & !(t\text{-}and \mid t\text{-}not \mid t\text{-}or \mid t\text{-}iff \mid t\text{-}imp \mid t\text{-}true \\ & \mid (not\text{-}t\,;\, t\text{-}or\,;\, (t\text{-}not \mid t\text{-}imp \mid t\text{-}iff \mid t\text{-}and \mid t\text{-}or)) \\ & \mid and\text{-}t \mid or\text{-}t \mid imp\text{-}t \mid iff\text{-}t \mid false\text{-}t) \end{aligned}$$

By applying this tactic exhaustively, we ensure that the consequent must be atomic, and the right-most antecedent must be atomic, or the negation of an atomic proposition. In this way, an almost normal form is achieved. If it denotes a tautology, this must be determined by application of the assumption rules.

The tactic *assum-tac* is defined in Section 8.1. Amongst other things, it attempts to apply the rule *AssumPred*; *iter-assum-tac* thins the goal repeatedly, applying *assum-tac* after each thinning.

$$iter\text{-}assum\text{-}tac := !(assum\text{-}tac \mid (thinr\text{-}tac1\,;\, iter\text{-}assum\text{-}tac))$$

In order to be ready for *iter-assum-tac*, the goal in normal form must be massaged, to remove any possible negations: all negated terms in the antecedent must be brought into the consequent (using *not-t*), and the resulting disjunctions simplified by use of *OrIl* and *OrIr*—using the angelic nondeterminism of alternation so that whichever disjunct is retained is the one which will match a term in the antecedent, via the assumption rules.

$$\begin{aligned} hyper\text{-}not &:= !(try\, hypershift\,;\, not\text{-}t) \\ clever\text{-}assum &:= exhaust\, hyper\text{-}not;; \\ &\quad\ exhaust\,(OrIl \mid OrIr); \\ &\quad\ (t\text{-}true \mid iter\text{-}assum\text{-}tac) \end{aligned}$$

hyper-not uses a tactic *hypershift* which brings the leftmost-possible antecedent to the right of the antecedent list. Informally, this can be defined as follows; a more general version could be defined by pattern-matching.

$$\begin{aligned} hypershift := {} & (lshift\,12 \mid lshift\,11 \mid lshift\,10 \mid \\ & lshift\,9 \mid lshift\,8 \mid lshift\,7 \mid lshift\,6 \mid \\ & lshift\,5 \mid lshift\,4 \mid lshift\,3 \mid lshift\,2) \end{aligned}$$

Over-all, then, the strategy of the tactic will be first to try and simplify the consequent and the right-most term of the antecedent, removing as many as possible of the resulting goals via the assumption rules. If there are any remaining goals, we will attempt to re-order the antecedent, and begin to simplify again. Re-ordering of the antecedent is conditional on the term thus selected not being atomic—that is, the shift is made only if it can be followed by a successful application of *props*.

$$\begin{aligned} \mathit{recurse} &:= !(\mathit{hypershift};\ \mathit{props})\ ;\mathit{prop\text{-}tac} \\ \mathit{prop\text{-}tac} &:= !(!\,\mathit{exhausts}\,\mathit{props};\\ &\qquad\qquad !(\mathit{tryclever\text{-}assum});\\ &\qquad\qquad \mathit{try}\,\mathit{recurse}) \end{aligned}$$

Using *prop-tac*, then, tautologies can be proved automatically.

For example, the commutativity of $\wedge$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P\ddagger Q \vdash Q}\ \textit{assum-tac}}{P \wedge Q \vdash Q}\ \textit{and-l} \quad \dfrac{\dfrac{}{P \vdash P}\ \textit{assum-tac}}{P\ddagger Q \vdash P}\ \textit{thinr}}{P \wedge Q \vdash Q \wedge P}\ \textit{and-r}}{\vdash P \wedge Q \Rightarrow Q \wedge P}\ \textit{Impl} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{Q\ddagger P \vdash P}\ \textit{assum-tac}}{Q \wedge P \vdash P}\ \textit{and-l} \quad \dfrac{\dfrac{}{Q \vdash Q}\ \textit{assum-tac}}{Q\ddagger P \vdash Q}\ \textit{thinr}}{Q \wedge P \vdash P \wedge Q}\ \textit{and-r}}{\vdash Q \wedge P \Rightarrow P \wedge Q}\ \textit{Impl}}{\vdash P \wedge Q \Rightarrow Q \wedge P \wedge Q \wedge P \Rightarrow P \wedge Q}\ \textit{AndI}}{\vdash P \wedge Q \Leftrightarrow Q \wedge P}\ \textit{IffDef}$$

The commutativity of $\vee$ is significantly different; it illustrates the use of *hyper-not*.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\, Q\ddagger P \vdash P}\ \textit{assum-tac}}{\neg\, Q\ddagger P \vdash P \vee Q}\ \textit{hyper-not} \quad \dfrac{\dfrac{}{Q \vdash Q}\ \textit{assum-tac}}{Q \vdash P \vee Q}\ \textit{OrIl}}{\neg\, Q\ddagger P \vee Q \vdash P}\ \textit{or-l}}{P \vee Q \ddagger \neg\, Q \vdash P}\ \textit{megashift}}{P \vee Q \vdash Q \vee P}\ \textit{t-or}}{\vdash P \vee Q \Rightarrow Q \vee P}\ \textit{t-imp} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\, P\ddagger Q \vdash Q}\ \textit{assum-tac}}{\neg\, P\ddagger Q \vee P \vdash Q}\ \textit{hyper-not} \quad \dfrac{\dfrac{}{P \vdash P}\ \textit{assum-tac}}{P \vee Q \vdash P}\ \textit{OrIl}}{Q \vee P \ddagger \neg\, P \vdash Q}\ \textit{or-l}}{Q \vee P \vdash P \vee Q}\ \textit{megashift}}{\vdash \wedge Q \vee P \vdash P \vee Q}\ \textit{t-or}}{\vdash P \vee Q \Rightarrow Q \vee P \wedge Q \vee P \vdash P \vee Q}\ \textit{t-imp}}{\vdash P \vee Q \Leftrightarrow Q \vee P}\ \textit{AndI}}{\vdash P \vee Q \Rightarrow Q \vee P \wedge Q \vee P \vdash P \vee Q}\ \textit{IffDef}$$

Of course, some further work is needed here. *prop-tac* is clearly not optimal, and we should prove that it terminates (easy, with the right bound function) and that it solves all tautologies (harder). To see that it is not optimal, consider the proof of $\vdash P \vee \neg P$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P \vdash P}\ \textit{assum-tac}}{\neg\, P\ddagger P \vdash \textit{false}}\ \textit{hyper-not}}{\neg\, P \vdash \neg\, P}\ \textit{t-not}}{\vdash \neg\, P \vee P}\ \textit{t-or}}{\neg\, \neg\, P \vdash P}\ \textit{not-l}}{\vdash P \vee \neg\, P}\ \textit{t-or}$$

Choice of the dual version of *t-or* would have rendered the initial goal immediately irreducible—and *assum-tac* would apply.

$$\dfrac{\dfrac{}{P \vdash P}\ \text{assum-ioc}}{\vdash P \vee \neg P}\ \text{i-or}'$$

However, such considerations are part of proof theory, and need not concern us here.

Part II

Expressions, and the Toolkit

Having used the propositional calculus as a vehicle for illustrating the style of proof used and the design of tactics, we are now in a position to consider extensions to the language that will provide the opportunity to do useful proofs.

One of the most important activities in proof is that of substitution. So given a theory about names and equality we can see how substitution works not only for predicates but also for declarations. This will allow schemas and other definitions to be expanded safely.

With the rules for generic definitions and expression constructs available to us we can prove properties about such objects as the empty set and the domain operator. The techniques used here illustrate how general properties of the objects defined in the toolkit can be proved using tactics. These tactics can then be re-applied whenever it is necessary to appeal to such properties in other proofs.

Contents

5 Paragraphs

The propositional calculus on its own is a barren language. In order to be able to use it to reason about real specifications, it is necessary to introduce expressions into the language. The given set declaration and the other means of definition provide a way of introducing new names. The membership and equality relations provide a way of writing propositions from expressions constructed from these new names.

5.1 Names and Scope

We use scope rules to define which names may be referred to at which point. The region of an expression (or even a specification) within which a variable can be referred to is called its scope. Z operates a system of nested scopes (Sennett 1987).

Each paragraph may use names defined in previous paragraphs and names introduced may be used in later paragraphs.When a variable is declared its scope extends to the end of the construct within which it was declared, except for any other sub-scopes within which the same name is re-declared.

We define the scope rules by giving a definition of the names introduced by paragraphs (alphabet) and the names used in them (free variables). The interaction between these two definitions defines the scope of variables.

Free Variables We define two different free variable functions: one for predicates, the other for expressions:[7]

$$\phi : Expr \rightarrow \mathbb{P}\, Name \quad ,$$
$$\Phi : Pred \rightarrow \mathbb{P}\, Name \quad .$$

The definition of these functions is given in the appendix.

Alphabet The alphabet function gives the set of names introduced by a paragraph, or sequence of paragraphs:

$$\alpha : Spec \rightarrow \mathbb{P}\, Name \quad .$$

The alphabets of the other simple paragraphs are the sets of names declared. The definition of this function for all paragraphs is given in the appendix.

Scope Term The scope introduction term $x^\bullet$ used in the rules below is not part of Z proper. It is used to denote the introduction of a new variable x and nothing more. It is a useful device for allowing paragraphs to be manipulated by

[7] When the free variables of schemas are calculated, we will need to be able to distinguish the two uses. We shall also calculate the free variables of paragraphs in the same way, interpreting them as expressions or predicates, as appropriate.

splitting them into ones which have no free variables and those that have no alphabet. The scope paragraph x^* has an alphabet containing just x:

$$\alpha x^* = \{x\} \quad .$$

Such a scoping paragraph has no free variables:

$$\phi x^* = \varnothing \quad .$$

The values of the set of free variables for other paragraphs are therefore derived from the free variables for their characteristic predicate.

5.2 Paragraph Rules

Given Sets A given set is a basic set from which others are constructed in a specification. The details of its membership are not given. [8] A given set introduction provides scoping information only:

$$\frac{x^* \vdash P}{[x] \vdash P} \; \mathit{GivenProp}$$

Definitions The definition $x := e$ introduces the new name x whose value is equal to the expression e. By using the notation x^* to indicate the introduction of the scope of the variable x we can explain the two declarations in terms of an introduction of scope and a constraining predicate:

$$\frac{\Gamma \dagger x^* \dagger x = e \vdash P \qquad \Gamma \vdash \mathrm{wf}(x := e)}{\Gamma \dagger x := e \vdash P} \; \mathit{DefProp}$$

The second sequent that must be satisfied $\Gamma \vdash \mathrm{wf}(x := e)$ is a condition that the name x is not used as a free variable in the expression e. In the event of such a condition not being satisfied, a judicious renaming will be needed to provide a meaning.

Declarations The declaration $x : s$ introduces a new name x whose value is contained in the set s. The rule for declaration follows the same structure as that for definition:

$$\frac{\Gamma \dagger x^* \dagger x \in s \vdash P \qquad \Gamma \vdash \mathrm{wf}(x : e)}{\Gamma \dagger x : s \vdash P} \; \mathit{DecProp}$$

Well-formedness Conditions Any name introduced which is already part of a specification is given a potentially new value. The side conditions on the declaration and definition rules given above guarantee that the new names introduced are not also free variables of the expressions used to define their values. The scoping rules make it impossible to have rules that use both the old and new

[8] No *a priori* assumptions are made about any internal structure which it might have. These sets can be empty, finite or infinite. Any further assumptions must be made explicit in the specification.

values of the variable in the same scope. This is a very common condition, so a special symbol **wf** Π will be used to state that the paragraph Π is well formed; its free variables are disjoint from its alphabet:

$$\mathbf{wf}\,\Pi \Leftrightarrow \phi(\Pi) \cap \alpha(\Pi) = \varnothing \quad .$$

5.3 New Structural Rules

Thinning The three types of declaration paragraph can be removed from the right only when the variable they declare is not a free variable of the predicate under consideration. The *Thinr* rule given for predicates is expanded to cater for the other forms of paragraph as follows:

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash \alpha\Pi \cap \Phi P = \varnothing}{\Gamma \ddagger \Pi \vdash P}\ \mathit{Thinr}$$

Swapping Two paragraphs can be swapped when there is no interaction between the names declared and their free variables. So the shift rule likewise is generalised as follows:

$$\frac{\Gamma_1 \ddagger \Pi_1 \ddagger \Gamma_2 \vdash P \quad \Gamma_1 \vdash \alpha\Pi_1 \cap \phi\Gamma_2 = \varnothing \quad \Gamma_1 \vdash \alpha\Gamma_2 \cap \phi\Pi_1 = \varnothing \quad \alpha\Pi_1 \cap \alpha\Gamma_2 = \varnothing}{\Gamma_1 \ddagger \Gamma_2 \ddagger \Pi_1 \vdash P}\ \mathit{Shift}$$

Rule Reversing The annotation $\uparrow\downarrow$ indicates that the rule can be applied in both directions—that is, the rule

$$\frac{\Gamma \vdash \Psi}{\Gamma' \vdash \Phi}\ \uparrow\downarrow$$

denotes both of the following inference rules

$$\frac{\Gamma \vdash \Psi}{\Gamma' \vdash \Phi} \quad \text{and} \quad \frac{\Gamma' \vdash \Phi}{\Gamma \vdash \Psi}$$

6 Substitution and Equality

Predicates An predicate to be evaluated under a substitution is often constructed during a proof. It usually has a temporary existence and is rarely used in specifications. However, rather than just give substitution rules that can be used to eliminate all occurrences of such a term, we shall give it a full meaning like all other terms.

$$\frac{\Gamma \ddagger x := e \vdash P}{\Gamma \vdash \{x := e\}P}\ \uparrow\downarrow$$

Expressions The notation for expression substitution is different to that for predicate substitution so as to prevent any parsing ambiguities when schemas are used as expressions and as predicates. The use of this notation can be seen in the Leibniz rule for paragraphs.

6.1 Leibniz

Leibniz' rule states that an expression e may be substituted in a predicate P for another expression u, providing that e and u are equal:

Leibniz' Rule Leibniz' rule can be derived from the properties of equality and substitution:

$$\frac{\Gamma \vdash (x := v) \odot P \qquad \Gamma \vdash e = v}{\Gamma \vdash (x := e) \odot P}\ \mathit{Leibniz}$$

Derivation of Rule We can derive this rule as follows:

$$\dfrac{\dfrac{\Gamma \ddagger e = u \qquad \dfrac{\dfrac{\Gamma \vdash (x := u) \odot P}{\Gamma \ddagger x^* \ddagger x = u \vdash P}}{\Gamma \ddagger x^* \ddagger x = e \ddagger e = u \vdash P}}{\Gamma \ddagger x^* \ddagger x = e \vdash P}}{\dfrac{\Gamma \ddagger x := e \vdash P}{\Gamma \vdash (x := e) \odot P}}$$

Alternate Versions In practice, a more sophisticate version of the rule is used. We normally find that we want to substitute one expression for another that is equal to it. In order to do this we need to construct an imaginary substitution instance and apply the rule of leibniz. For example if the predicate P could be rewritten as $(x := e) \odot P'$ then we could apply the rule of leibniz to the following sequent:

$$\Gamma \ddagger e = u \vdash P$$

to give us the following:

$$\Gamma \vdash (x := u) \odot P'$$

and applying the substitution we have the following derived rule:

$$\frac{\Gamma \vdash Q}{\Gamma \ddagger e = u \vdash P}\ \mathit{leibniz}$$

where for some predicate R and variable x:

$$\begin{aligned} P &\equiv (x := e) \odot R \\ Q &\equiv (x := u) \odot R \end{aligned}$$

This derived rule can be seen in practice in the proof of the property of *AddBirthday* given earlier:

$$\frac{\cdots \vdash \mathrm{dom}(\mathit{birthday} \cup \{(\mathit{name}?, \mathit{date}?)\}) = \mathit{known} \cup \{\mathit{name}?\}}{\ddagger \mathit{birthday}' = \mathit{birthday} \cup \{(\mathit{name}?, \mathit{date}?)\} \vdash \mathrm{dom}\, \mathit{birthday}' = \mathit{known} \cup \{\mathit{name}?\}}\ \mathit{leibniz}$$

6.2 Leibniz for Paragraphs

We can substitute equals for equals within paragraphs in the antecedent, following the rule of Leibniz. This again is a derived rule:

$$\frac{\Gamma \ddagger \{ x := v \} \circ \Pi \vdash P \qquad \Gamma \vdash e = v}{\Gamma \ddagger \{ x := e \} \circ \Pi \vdash P} \; P\text{-}Leib$$

A useful version of this rule for expanding schema definitions can be derived using *DefProp*:

$$\frac{T \vdash P}{S := T \ddagger S \vdash P} \; x\text{-}sch\text{-}leib\text{-}t$$

In the introduction, we postulated a theorem about *AddBirthday*:

$$AddBirthday \vdash known' = known \cup \{name?\}$$

We can expand the definition of *AddBirthday* by applying a tactic constructed from *S-Exp* followed by *thinl* (to remove the definition once applied) as follows:

$$\frac{\Delta Birthday;\ name? : N;\ date? : D \mid \cdots \vdash k' = k \cup \{name?\}}{AddBirthday := \Delta Birthday;\ name? : N;\ date? : D \mid \cdots \ddagger AddBirthday \vdash k' = k \cup \{name?\}} \; x\text{-}sch\text{-}leib\text{-}t$$

6.3 Equational Reasoning

In order to prove the property of *AddBrithday*, Spivey uses an equational reasoning style. He transforms the left hand side of an equality into the right by substituting expressions for other equal expressions. The proof is presented using informal justifications about these equalities. Here we shall follow the equational style, but also provide a framework in which the justifications can be discharged.

To set up the equational proof in our sequent style we arrange that each step involves the *cut* of a lemma (generally some equality), and uses leibniz to rewrite the current goal according to the equation that has been introduced.

$$\dfrac{\dfrac{}{\Gamma \vdash u = v}\; t1 \qquad \dfrac{\dfrac{}{\Gamma \vdash w = x}\; t2 \qquad \dfrac{\vdots}{\Gamma \vdash A = D}}{\Gamma \vdash A = C}\; cut\text{-}tac;\ leibniz}{\Gamma \vdash A = B}\; cut\text{-}tac;\ leibniz$$

The general tactic In this way, many tree-like proofs can be reduced to an essentially linear form. Each equational reasoning step is accomplished by a tactic *eqtac*, which cuts in the predicate p, proves it using the supplied tactic t, and uses it to rewrite the goal.

$$eqtac\ t\ p := cut\text{-}tac\ p;\ (t \parallel leibniz)$$

To make presentation of such trees easier we move the justification into the proviso, to produce a more vertical format:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash A = D}}{\Gamma \vdash A = C}\ w = x}{\Gamma \vdash A = B}\ u = v$$

The over-all *AddBirthday* proof can be set up equationally, following the structure of Spivey's proof. The lemmas will be discussed below.

$$\begin{aligned}
&\mathit{spivey}\text{-}5 := \mathit{eqtac}(\mathit{lemma}1)\\
&\qquad (\mathit{known}' = \mathrm{dom}\,\mathit{birthday}');\\
&\quad \mathit{eqtac}(\mathit{lemma}2)\\
&\qquad (\mathit{birthday}' = \mathit{birthday} \cup \{(\mathit{name}?, \mathit{date}?)\});\\
&\quad \mathit{eqtac}(\mathit{lemma}3)\\
&\qquad (\mathrm{dom}(\mathit{birthday} \cup \{(\mathit{name}?, \mathit{date}?)\}) =\\
&\qquad\qquad (\mathrm{dom}\,\mathit{birthday}) \cup (\mathrm{dom}\{(\mathit{name}?, \mathit{date}?)\}));\\
&\quad \mathit{eqtac}(\mathit{lemma}4)\\
&\qquad (\mathrm{dom}\{(\mathit{name}?, \mathit{date}?)\} = \{\mathit{name}?\});\\
&\quad \mathit{eqtac}(\mathit{lemma}5)\\
&\qquad (\mathrm{dom}\,\mathit{birthday} = \mathit{known});\\
&\quad \mathit{Refl}
\end{aligned}$$

This tactic can generate the following tree

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{AB \vdash k \cup \{n?\} = k \cup \{n?\}}\ \mathit{Refl}}{AB \vdash k \cup \{n?\} = \mathrm{dom}\, b \cup \{n?\}}\ \mathrm{dom}\, b = k}{AB \vdash k \cup \{n?\} = \mathrm{dom}\, b \cup \mathrm{dom}\{n? \mapsto d?\}}\ \mathrm{dom}\{n? \mapsto d?\} = \{n?\}}{AB \vdash k \cup \{n?\} = \mathrm{dom}\, b \cup \{n? \mapsto d?\}}\ \mathrm{dom}\, b \cup \{n? \mapsto d?\} = \mathrm{dom}\, b \cup \mathrm{dom}\{n? \mapsto d?\}}{AB \vdash k \cup \{n?\} = \mathrm{dom}\, b'}\ b' = b \cup \{n? \mapsto d?\}}{AB \vdash k \cup \{n?\} = k'}\ k' = \mathrm{dom}\, b'$$

7 Quantification

There are four quantifier rules presented in figure 3. The first rule, which gives an introduction and elimination procedure for universal quantification, is sufficient to define the other three (assuming the de Morgan correspondence between universal and existential quantification).

Free Variables Quantified predicates introduce a new scope. In the universal quantification $\forall x : s \bullet P$, the scope for the variable x is the predicate P, so if x is a free variable of the predicate P, it is captured. The free variables of the quantification are calculated as follows:

$$\Phi \forall x : s \bullet P = \phi s \cup (\Phi P \setminus \{x\})$$

The free variables of the other quantifiers are calculated in the same way, and the definitions of them are given in the appendix.

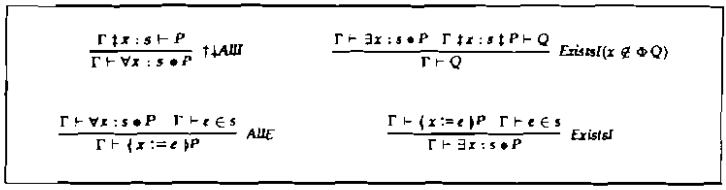


Figure 3: The Quantifier Rules

Substitution The result of applying a substitution to a quantified predicate depends on whether the substitution can be performed without capture, and whether the variable being replaced is bound by the quantifier.

The only occurrences of the variable x that may be substituted are the free occurrences. So if a component of the substitution is bound by the quantifier, then only the free variables of that component in the declarations will change.

$$\{y := v\}(\forall y : s \bullet P) \quad \equiv \quad \forall y : \{y := v\} \ominus s \bullet P$$

$$\{x := v\}(\forall y : s \bullet P) \quad \equiv \quad \forall y : \{x := v\} \ominus s \bullet \{x := v\} P \quad \text{where } y \notin \phi v.$$

7.1 Renaming

In the textual evaluation of the substitution $\{x := e\} P$ the free variables of e stand in danger of being bound in P. So the substitution rules have a side condition to prevent variable capture. When a clash occurs, renaming can take place to avoid unwanted variable capture.

In the predicate $\forall x : s \bullet P$ the variable x is said to be bound. This name is not significant and can be systematically replaced in the predicate without any change in meaning. It acts as a place holder. Syntactic renaming is safe only when it does not capture any new free variables.

$$\frac{\vdash \forall y : s \bullet \{x := y\} P}{\vdash \forall x : s \bullet \{y := x\} P} \uparrow\downarrow$$

$$\frac{\dfrac{\dfrac{\dfrac{\vdash \forall y : s \bullet \{x := y\} P}{y : s \ddagger x := y \vdash P}}{x : s \ddagger y := x \vdash P}}{x : s \vdash \{y := x\} P}}{\vdash \forall x : s \bullet \{y := x\} P}$$

So for $y \notin \Phi P$ we have

$$\forall x : s \bullet P \equiv \forall y : s \bullet \{x := y\} P \quad .$$

Similar rules apply for set comprehension and existential quantification.

7.2 Tactics for Quantifiers

When a universal quantifier is encountered in the antecedent, can be removed by supplying a binding which is suitable to specialize it for the task at hand.

$$\begin{aligned} &\mathit{all\text{-}tb} := \pi\, S, p, q \bullet \forall S \bullet p \vdash q \rightarrow \\ &\qquad \mathit{cut\text{-}tac}(b \in S);\ (\mathbf{skip} \parallel \mathit{lshift}\, 2;\\ &\qquad \mathit{cut\text{-}tac}(b \odot p);\ (\mathit{AllES}\ ;\ \mathit{iter\text{-}assum\text{-}tac} \parallel \mathbf{skip})) \end{aligned}$$

$$\dfrac{\dfrac{\dfrac{}{b \in S \natural \forall S \bullet p \vdash \forall S \bullet p}\ \mathit{assum\text{-}tac} \quad \dfrac{}{b \in S \natural \forall S \bullet p \vdash b \in S}\ \mathit{iter\text{-}assum\text{-}tac}}{b \in S \natural \forall S \bullet p \vdash b \odot p}\ \mathit{AllE} \qquad b \in S \natural \forall S \bullet p \natural b \odot p \vdash q}{\forall S \bullet p \vdash b \in S \qquad b \in S \natural \forall S \bullet p \vdash q}\ \mathit{cut\text{-}tac}}{\forall S \bullet p \vdash q}\ \mathit{cut\text{-}tac}$$

For symmetry, *t-all* is defined (as *AllI*). Likewise, we define *exists-t*, which simply converts the quantified predicate into a schema paragraph and predicate:

$$\begin{aligned} &\mathit{exists\text{-}t} := \pi\, S, p, q \bullet \exists S \bullet p \vdash q \rightarrow \\ &\qquad \mathit{ExistsESp};\ (\mathit{assum\text{-}tac} \parallel \mathbf{skip}) \end{aligned}$$

$$\dfrac{\dfrac{}{\exists S \bullet p \vdash \exists S \bullet p}\ \mathit{assum\text{-}tac} \qquad s \natural p \vdash q}{\exists S \bullet p \vdash q}\ \mathit{ExistsE}$$

A tactic *t-exists* can be defined (using *ExistsI*) in a way analogous to *all-t*—i.e. it takes as a parameter a binding which is to be used to provide a witness for the existential quantification. Frequently, however (especially when the rule of set comprehension is used—see below), the 'one-point' rule is useful in this situation:

$$\begin{aligned} &\mathit{t\text{-}onept} := \pi\, S, e, x \bullet \vdash \exists S \bullet e = x \rightarrow \\ &\qquad \mathit{ExistsI}(\!| x := e |\!);\ (\mathit{subst\text{-}tac};\ \mathit{Refl} \parallel \mathbf{skip}) \end{aligned}$$

$$\dfrac{\dfrac{\dfrac{}{\vdash e = e}\ \mathit{Refl}}{\vdash (\!| x := e |\!) \odot (e = x)}\ \mathit{subst\text{-}tac} \qquad \vdash (\!| x := e |\!) \in S}{\vdash \exists S \bullet e = x}\ \mathit{ExistsI}$$

8 Derived Structural Rules

8.1 Generalized Assumption

The new sorts of paragraphs introduced in Section 5 give rise to some additional *assumption* rules. By the use of the scoping term, these can be derived rules, but they are presented as primitive in the appendix.

$$\frac{}{x := e \vdash x = e}\ \mathit{AssumDefin}(\mathbf{wf}(x := e))$$

$$\frac{}{x : s \vdash x \in s}\ \mathit{AssumDecl}(\mathbf{wf}(x : s)) \qquad \frac{}{S \vdash S}\ \mathit{SchemaAss}(\mathbf{wf}(S))$$

(In the last rule, the schema is playing the rôle of a predicate in the consequent, and a declaration in the antecedent. Schemas will be discussed more fully in Sections 11–13). These assumption rules are collected together into the tactic *assum-tac*

$$\mathit{assum\text{-}tac} := \mathit{AssumPred} \mid \mathit{AssumDefin} \mid \mathit{Assumdecl} \mid \mathit{SchemaAss}$$

We have already seen the definition of *iter-assum-tac*, which thins the goal repeatedly, applying *assum-tac* after each thinning (see Section 4).

8.2 *par-pred-t*

It is sometimes desirable to extract the predicate component of a paragraph without splitting it into a scoping term/predicate pair.[9] The tactic *par-pred-t* uses the assumption rules to copy the paragraph as a predicate.

$$\begin{aligned}\mathit{par\text{-}pred\text{-}t} := {} & (\pi x, e, Q \bullet x := e \vdash Q \rightarrow \mathit{cut\text{-}tac}(x = e) \mid \\ & \pi x, e, Q \bullet x : e \vdash Q \rightarrow \mathit{cut\text{-}tac}(x \in e) \mid \\ & \pi S \bullet S \vdash Q \rightarrow \mathit{cut\text{-}tac}(S)); \\ & (\mathit{assum\text{-}tac} \parallel \mathbf{skip})\end{aligned}$$

An example of the application of *par-pred-t*:

$$\frac{\Gamma \upharpoonright x : e \upharpoonright x \in e \vdash P}{\Gamma \upharpoonright x : e \vdash P}\ \mathit{par\text{-}pred\text{-}t}$$

8.3 *up-down-tac*

Many of the rules for expressions carry the $\uparrow\downarrow$ annotation. Whilst the downwards instance of the rule is not generally useful in its own right (as it creates complex terms from simpler ones), it can be used to apply a similar inference in the antecedent. As this will be a common pattern of reasoning, we have a tactic *up-down-tac*, which accomplishes this.

For example, for the inference rule

$$\frac{\Gamma \vdash P}{\Gamma \vdash Q}\ \uparrow\downarrow r$$

[9] One example is when the paragraph declares a schema. By splitting it up, we loose the ability to calculate the schema's alphabet.

may be applied on the left as follows:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \ddagger Q \vdash Q}\ \textit{assum-tac}}{\Gamma \ddagger Q \vdash P}\ r \qquad \dfrac{\dfrac{\Gamma \ddagger P \vdash R}{\Gamma \ddagger Q \ddagger P \vdash R}\ \textit{drop-snd} \qquad \dfrac{}{\Gamma \ddagger Q \ddagger R \vdash R}\ \textit{assum-tac}}{\Gamma \ddagger Q \vdash P \vee R}\ \textit{OrIr}}{\Gamma \ddagger Q \vdash R}\ \textit{OrEPR}$$

The tactic *up-down-tac* determines the parameters for *OrE* by pattern-matching, so that we have:

$$\dfrac{\Gamma \vdash P}{\Gamma \vdash Q}\ \uparrow\downarrow r \qquad\qquad \dfrac{\Gamma \ddagger Q \vdash R}{\Gamma \ddagger P \vdash R}\ \textit{up-down-tac}\ r$$

This means can be used to define some tactics which operate on terms in the antecedent. Again, their application will frequently be combined with simple proposition/predicate calculus rules. Moreover, it is often valuable to be able to apply the antecedent tactics in a single step to declarations and definitions, as well as membership and equality predicates. Therefore many tactics are preceded by an optional *par-pred-t*, which converts paragraphs to predicates where necessary.

8.4 Apply-inwards

We have given various rules which act on the antecedent, but in general they work only on the leftmost paragraph of the antecedent. The rule of *shift* can be use to reorder the terms in the antecedent, but often it is useful to leave the order unchanged, applying a rule or tactic to one of the internal paragraphs.

One way to accomplish this is to make use of a number of reversible rules which take antecedent paragraphs, and make them into part of the consequent. We have already encountered these rules:

$$\dfrac{\Gamma \ddagger S \vdash P}{\Gamma \vdash \forall S \bullet P}\ \uparrow\downarrow \textit{AllI} \qquad \dfrac{\Gamma \ddagger b \vdash P}{\Gamma \vdash b \odot P}\ \uparrow\downarrow \textit{UseBind} \qquad \dfrac{\Gamma \ddagger P \vdash Q}{\Gamma \vdash P \Rightarrow Q}\ \uparrow\downarrow \textit{IffI}$$

Tactics can be used to apply whichever of these rules is applicable, to move terms from left to right, or right to left.

$$\textit{left-right} := \textit{AllI} \uparrow \mid \textit{UseBind} \uparrow \mid \textit{IffI} \uparrow$$

$$\textit{right-left} := \textit{AllI} \downarrow \mid \textit{UseBind} \downarrow \mid \textit{IffI} \downarrow$$

By repeatedly applying these rules, we may apply a chosen tactic arbitrarily far inside an antecedent.

$$\textit{apply-inwards}\ n\ t := \textit{repeat-tac}\ n\ \textit{left-right}\ ;\ t\ ;\ \textit{repeat-tac}\ n\ \textit{right-left}$$

We will generally denote *apply-inwards n t* by $\curvearrowright_n t$, omitting the n when just one inwards movement is needed. For example,

$$\frac{\Gamma \ddagger \forall x : t \bullet x \in s \ddagger T \ddagger U \vdash P}{\Gamma \ddagger t \in \mathbb{P} s \ddagger T \ddagger U \vdash P} \; \mathcal{D}(up\text{-}down\text{-}tac\ Powerset\ x)$$

9 Expressions

In Z, we are able to discuss a variety of expressions—sets, cartesian products, labelled products (schema bindings), and functions. Inference rules are provided to permit these expressions to be simplified, and expressed in terms of one another. The basic rules are given in the appendix.

In general, however, those basic rules make steps which are unnecessarily small, so here we discuss some tactics offering derived rules which are more generally useful.

9.1 Expressions in the Consequent

Set Comprehension The rule for set comprehension converts a comprehension into an existential quantification. In the event that the comprehension declares exactly one variable, the one-point rule can be used to simplify the resulting predicate.

t-setcomp := *Setcomp* ; *t-one-pt*

$$\frac{\vdash x \in s \quad \vdash \{ y := x \} \odot P}{\vdash x \in \{ y : s \mid P \bullet y \}} \; t\text{-}setcomp$$

Powerset The powerset rule will always be followed immediately by *AllI*.

t-powerset := *Powerset x*; *AllI*

$$\frac{x : e \vdash x \in u}{\vdash e \in \mathbb{P} u} \; t\text{-}powerset$$

Prodmem

t-prodmem := *Prodmem*; *exhaustsAndI*

$$\frac{\vdash u.1 \in s_1 \quad \cdots \quad \vdash u.n \in s_n}{\vdash u \in s_1 \times \cdots \times s_n} \; t\text{-}prodmem$$

We will use a large number of such derived rules, without giving all the definitions.

9.2 Expressions in the Antecedent

Whilst the rules for expressions are all expressed using terms in the consequent, the tactic *up-down-tac* previously presented can be used to apply then equally well in the antecedent.

Set Comprehension Set comprehension in the antecedent can be followed by *exists-t*, to remove the existential quantifier.

$$\begin{array}{l} \mathit{setcomp\text{-}t} := !(\mathbf{skip} \mid \mathit{par\text{-}pred\text{-}t}); \\ \qquad \mathit{up\text{-}down\text{-}tac}\ \mathit{Setcomp};\ \mathit{exists\text{-}t} \end{array}$$

$$\frac{\Gamma \ddagger S \ddagger e = u \vdash P}{\Gamma \ddagger e \in \{S \bullet u\} \vdash P}\ \mathit{setcomp\text{-}t} \quad \text{and also} \quad \frac{\Gamma \ddagger e : \{S \bullet u\} \ddagger S \ddagger e = u \vdash P}{\Gamma \ddagger e : \{S \bullet u\} \vdash P}\ \mathit{setcomp\text{-}t}$$

9.3 Larger derived Expression Rules

Singleton in powerset

$$\mathit{sing\text{-}power} := \mathit{t\text{-}powerset};\ \mathit{extmem\text{-}t};\ \mathit{t\text{-}leibniz};\ \mathit{thinr\text{-}tac}\,2$$

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\vdash a \in A}}{x_1 : \{a\} \ddagger x_1 = a \vdash a \in A}}{x_1 : \{a\} \ddagger x_1 = a \vdash x_1 \in A}}{x_1 : \{a\} \vdash x_1 \in A}}{\vdash \{a\} \in \mathbb{P} A}$$

Tuple in Product

$$\begin{array}{l} \mathit{onesel} := \pi\, a, b, A \bullet \vdash (a,b).1 \in A \rightarrow \\ \qquad \mathit{cut\text{-}tac}((a,b).1 = a);\ (\mathit{t\text{-}tupleequ};\ \mathit{Refl} \\ \qquad\quad \| x\text{-}\mathit{gen\text{-}t\text{-}leibniz};\ \mathit{thinr\text{-}tac}\,1) \end{array}$$

$$\cfrac{\cfrac{\cfrac{}{\vdash a = a}\ \mathit{Refl}}{\vdash (a,b).1 = a}\ \mathit{t\text{-}tupleequ} \quad \cfrac{\cfrac{}{\vdash a \in A}\ \mathit{thinr}}{(a,b).1 = a \vdash (a,b).1 \in A}\ \mathit{t\text{-}leibniz}}{\vdash (a,b).1 \in A}\ \mathit{cut\text{-}tac}$$

$$\mathit{tuple\text{-}in\text{-}prod} := \mathit{t\text{-}prodmem};\ (\mathit{onesel} \parallel \mathit{twosel})$$

$$\cfrac{\cfrac{\vdash a \in A}{\vdash (a,b).1 \in A}\ \mathit{onesel} \quad \cfrac{\vdash b \in B}{\vdash (a,b).2 \in B}\ \mathit{twosel}}{\vdash (a,b) \in A \times B}\ \mathit{t\text{-}prodmem}$$

Singleton in Power Product

pair-pow-prod-mem := *sing-power*; *tuple-in-prod*

$$\dfrac{\dfrac{\vdash a \in A \quad \vdash b \in B}{\vdash (a,b) \in A \times B}\ \textit{tuple-in-prod}}{\vdash \{(a,b)\} \in \mathbb{P}(A \times B)}\ \textit{sing-power}$$

9.4 Using The Toolkit

9.4.1 Related Definitions

Whenever a predicate appears in the consequent requiring the proof that a certain term belongs to a partial function space, we will generally need to invoke the definition of partial functions.

pfun-is-rel-and-fun := *t-leibniz* ; *t-setcomp* ; *SchemaMem* ; *subst-tac* ; *t-and*;
(*BindProd* ; *subst-tac*; *thinr-tac* 1 ‖ *thinr-tac* 2)

$$\dfrac{\begin{array}{l}\vdash x \in \{f : X \leftrightarrow Y \mid \forall x_1 : X; y_1, y_2 : Y \bullet \\ \qquad (x_1, y_1) \in f \land (x_1, y_2) \in f \Rightarrow y_1 = y_2\}\end{array}}{\begin{array}{l}\vdash x \in X \leftrightarrow Y \\ X \pfun Y = \{f : X \leftrightarrow Y \mid \forall x_1 : X; y_1, y_2 : Y \bullet \\ \qquad (x_1, y_1) \in f \land (x_1, y_2) \in f \Rightarrow y_1 = y_2\} \\ \vdash \\ x \in X \pfun Y\end{array}}\ \textit{pfun-is-rel-and-fun}$$

Likewise, membership of a relation space can be reduced to membership of the powerset of a cartesian product.

rel-is-power := *t-leibniz*; *thinr-tac* 1

$$\dfrac{\vdash x \in \mathbb{P}(X \times Y)}{X \leftrightarrow Y = \mathbb{P}(X \times Y) \vdash x \in X \leftrightarrow Y}\ \textit{rel-is-power}$$

9.4.2 Harwood's Theorem

In order to illustrate the use of some of these rules about expressions, we present a proof of a simple property derived from the toolkit definitions. It has been called 'Harwood's Theorem'.

$$\mathrm{dom}\{(a,b)\} = \{a\}$$

To make this amenable to proof, we state it, suitably quantified, together with the relevant definitions, in a sequent:

$$
\begin{array}{l}
NAME \leftrightarrow DATE = \mathbb{P}(NAME \times DATE)\ddagger \\
\forall R : NAME \leftrightarrow DATE \bullet \mathrm{dom}\, R = \\
\qquad \{x_3 : NAME;\ y : DATE \mid (x_3, y) \in R \bullet x_3\} \\
\vdash \\
\forall a : NAME;\ b : DATE \bullet \mathrm{dom}\{(a,b)\} = \{a\}
\end{array}
$$

The first step in the proof is to strip off the quantifier, and since this produces a declaration in the antecedent, the next stage is to extract the predicate content of that declaration.

unquantify := *t-all*: *sand-t*; *bindprodp*; *apply-inwards*(1)*bindprodp*

Giving

$$
\begin{array}{l}
NAME \leftrightarrow DATE = \mathbb{P}NAME \times DATE\ddagger \\
\forall R : NAME \leftrightarrow DATE \bullet \mathrm{dom}\, R = \\
\qquad \{x_3 : NAME;\ y : DATE \mid (x_3, y) \in R \bullet x_3\}\ddagger \\
a : NAME \wedge b : DATE\ddagger \\
a \in NAME\ddagger \\
b \in DATE \\
\vdash \\
\mathrm{dom}\,\{(a,b)\} = \{a\}
\end{array}
$$

Next, the definition of dom must be specialized for this particular instance—it is brought to the right-hand end of the antecedent, and then ***all-t*** is applied.

instantiate-definition := *lshift*4; *all-t*($R := \{(a,b)\}$)

This gives two subgoals: one to prove that the supplied singleton is indeed a relation; the other to use the property of the definition to prove the main goal.

$$
\begin{array}{l}
NAME \leftrightarrow DATE = \mathbb{P}NAME \times DATE\ddagger \\
a : NAME \wedge b : DATE\ddagger \\
a \in NAME\ddagger \\
b \in DATE\ddagger \\
\forall R : NAME \leftrightarrow DATE \bullet \mathrm{dom}\, R = \\
\qquad \{x_3 : NAME;\ y : DATE \mid (x_3, y) \in R \bullet x_3\} \\
\vdash \\
(\ R := \{(a,b)\}\ ) \in [R : NAME \leftrightarrow DATE]
\end{array}
$$

$$
\begin{array}{l}
NAME \leftrightarrow DATE = \mathbb{P}\, NAME \times DATE\ddagger \\
a : NAME \wedge b : DATE\ddagger \\
a \in NAME\ddagger \\
b \in DATE\ddagger \\
(\, R := \{(a,b)\}\,) \in [R : NAME \leftrightarrow DATE]\ddagger \\
\forall R : NAME \leftrightarrow DATE \bullet \mathrm{dom}\, R = \\
\qquad \{x_3 : NAME;\ y : DATE \mid (x_3, y) \in R \bullet x_3\}\ddagger \\
(\, R := \{(a,b)\}\,) \odot \mathrm{dom}\, R = \\
\qquad \{x_3 : NAME;\ y : DATE \mid (x_3, y) \in R \bullet x_3\} \\
\vdash \\
\mathrm{dom}\, \{(a,b)\} = \{a\}
\end{array}
$$

First Subgoal The first subgoal is approached by rewriting the consequent using the rule schema binding membership, discarding the definition of dom, and rewriting using the definition of $\leftrightarrow$.

binding-suits-decl := *BindProd*; *subst-tac*; *thinr-tac* 1;
lshift 4; *x-gen-t-leibniz*; *thinr-tac* 1

$$
\begin{array}{l}
a : NAME \wedge b : DATE\ddagger \\
a \in NAME\ddagger \\
b \in DATE \\
\vdash \\
\{(a,b)\} \in \mathbb{P}(NAME \times DATE)
\end{array}
$$

This is finally completed by appealing to the tactic *pair-pow-prod-mem* defined above, followed by *iter-assum-tac*.

Second Subgoal First, the substitution introduced by *all-t* must be made, and then the transitivity rule used to rewrite the consequent. The information about dom can then be thinned.

use-instantiation := *subst-tac*; *trans-tac*; *thinr-tac* 3

$$
\begin{array}{l}
a : NAME \wedge b : DATE\ddagger \\
a \in NAME\ddagger \\
b \in DATE \\
\vdash \\
\{a\} = \{x_3 : NAME;\ y : DATE \mid (x_3, y) \in \{(a,b)\} \bullet x_3\}
\end{array}
$$

The resulting goal is an equality. To prove that the two terms are equal, we use the rule of extension, modified by a tactic to remove the quantification, giving two subgoals.

$$\frac{x : s \vdash x \in t \quad x : t \vdash x \in s}{\vdash s = t}\ t\text{-}seteq$$

$a : NAME \wedge b : DATE\ddagger$
$a \in NAME\ddagger$
$b \in DATE\ddagger$
$x : \{a\}$
$\vdash$
$x \in \{x_3 : NAME;\ y : DATE \mid (x_3, y) \in \{(a,b)\} \bullet x_3\}$

$a : NAME \wedge b : DATE\ddagger$
$a \in NAME\ddagger$
$b \in DATE\ddagger$
$x : \{x_3 : NAME;\ y : DATE \mid (x_3, y) \in \{(a,b)\} \bullet x_3\}$
$\vdash$
$x \in \{a\}$

Simplification of the first of these subgoals entails rewriting the singleton expresison in the antecedent, and using this to simplify the comprehension in the consequent, before applying the rule of set comprehension. After this, a simpleapplication of the one-point rule does not suffice, since a value must also be supplied for y. Once again, this gives two goals; one to prove that the supplied binding belongs to the schema part of the set comprehension, the other to using that binding to simplify the comprehension. These are readily discharged; one by substitution and reflection; the other by schema membership, schema conjunction, and extension.

$$\begin{aligned} \textit{singleton-mem-t} := {} & \textit{par-pred-t};\ \textit{extmem-t};\ \textit{x-gen-t-leibniz}; \\ & \textit{thinr-tac}\,2;\ \textit{x-t-setcomp}; \\ & (\textit{subst-tac};\ \textit{Refl} \parallel \textbf{skip}); \\ & \textit{t-schmem};\ (\textit{t-sandh} \ ; \textit{iter-assum-tac} \\ & \parallel \textit{t-extmem};\ \textit{Refl}) \end{aligned}$$

$$\textit{x-t-setcomp} := \textit{Setcomp};\ \textit{t-exists}\{\, x_3 := a, y := b \,\}$$

The final remaining goal is solved by a broadly symmetric tactic. Extension membership is applied in the consequent, and set comprehension in the antecedent. Rules for tuple selection and equality complete the proof.

$$\begin{aligned} \textit{t-singleton-mem} := {} & \textit{t-extmem};\ \textit{setcomp-t};\ \textit{trans-tac};\ \textit{thinr-tac}\,1; \\ & \textit{drop-snd};\ \textit{drop-snd};\ \textit{sconstrddag-t};\ \textit{extmem-t}; \\ & \textit{up-down-tacTupleSel};\ \textit{and-t};\ \textit{thinr-tac}\,1; \\ & \textit{trans-tac};\ \textit{t-tupleequ};\ \textit{Refl} \end{aligned}$$

Collecting these parts together, then, we have

$$
\begin{aligned}
\mathit{harwood\text{-}tac} := {} & \mathit{unquantify};\ \mathit{instantiate\text{-}definition};\\
& (\mathit{binding\text{-}suits\text{-}decl};\ \mathit{pair\text{-}pow\text{-}prod\text{-}mem};\ \mathit{iter\text{-}assum\text{-}tac}\\
& \|\mathit{use\text{-}instantiation};\ \mathit{t\text{-}seteq};\\
& \quad (\mathit{singleton\text{-}mem\text{-}t} \| \mathit{t\text{-}singleton\text{-}mem}))
\end{aligned}
$$

10 Generics

The uses of toolkit definitions given above have overlooked the fact that the usual definitions are stated using generics, not with ready-supplied parameters suitable to the problem at hand.

In order to specialize a generic abbreviation for a particular instantiation, we use the rule *GenSpec*.

$$\frac{\Gamma \ddagger x[y] := e \vdash \{ y := u \}_{\odot} e \in v}{\Gamma \ddagger x[y] := e \vdash x[u] \in v}\ \mathit{GenSpec}$$

A similar rule is provided for generic paragraphs (see the appendix), but its use entails slightly more work if the paragraph does not uniquely define the names it introduces.

Using this rule, we can define a tactic which implements a version of the *assumption* rules for generic terms.

$$
\begin{aligned}
\mathit{genspec\text{-}assum} := {} & \pi\, n, X, t, e, u \bullet n[X] := t \vdash n[e] = u \longrightarrow\\
& \mathit{cut\text{-}tac}(n[e] \in \{z : \mathbb{P} u \mid z = u\});\\
& (\mathit{GenSpec};\ \mathit{t\text{-}setcomp};\ \mathit{t\text{-}schemamem};\ \mathit{t\text{-}and}\\
& \quad (\mathit{t\text{-}bindprod};\ \mathit{t\text{-}powerset};\ \mathit{assum\text{-}tac}\\
& \quad \|\mathit{subst\text{-}tac};\ \mathit{Refl})\\
& \|\\
& \mathit{setcomp\text{-}t};\ \mathit{trans\text{-}tac};\ \mathit{thinr\text{-}tac}\,1;\\
& \mathit{SConstrDdag};\ \mathit{symm};\ \mathit{assum\text{-}tac})
\end{aligned}
$$

This tactic gives the following proof tree, provided $\{ X := e \}_{\odot} t$ and u are identical.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{z : \{ X := e \}_{\odot} t \vdash z \in u}\ \mathit{assum\text{-}tac}
}{\vdash \{ X := e \}_{\odot} t \in \mathbb{P} u}\ \mathit{t\text{-}powerset}
}{\vdash \{ \{ z := \{ X := e \}_{\odot} t \} \in [z : \mathbb{P} u]}\ \mathit{t\text{-}bindprod}
\qquad
\dfrac{
\dfrac{}{\vdash u = \{ X := e \}_{\odot} t}\ \mathit{Refl}
}{\vdash \{ \{ z := \{ X := e \}_{\odot} t \} \odot (u = z)}\ \mathit{subst}
}{\vdash \{ \{ z := \{ X := e \}_{\odot} t \} \in [z : \mathbb{P} u] \wedge (u := z) \odot (u = z)}\ \mathit{t\text{-}and}
}{\vdash \{ z := \{ X := e \}_{\odot} t \} \in [z : \mathbb{P} u \mid u = z]}\ \mathit{t\text{-}schemamem}
}{\vdash \{ X := e \}_{\odot} t \in \{z : \mathbb{P} u \mid z = u\}}\ \mathit{t\text{-}setcomp}
}{n[X] := t \vdash n[e] \in \{z : \mathbb{P} u \mid z = u\}}\ \mathit{GenSpec};\ \mathit{subst}
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\ldots \ddagger z : \mathbb{P} u \ddagger z = u \vdash u = z}\ \mathit{symm};\ \mathit{assum\text{-}tac}
}{\ddagger z : \mathbb{P} u \mid z = u \vdash u = z}\ \mathit{SConstrDdag};\ \mathit{symm}
}{\ldots \ddagger z : \mathbb{P} u \mid z = u \ddagger n[e] = z \vdash n[e] = u}\ \mathit{trans\text{-}tac};\ \mathit{thinr}
}{\ldots \ddagger n[e] \in \{z : \mathbb{P} u \mid z = u\} \vdash n[e] = u}\ \mathit{setcomp\text{-}t}
}{n[X] := t \vdash n[e] = u}\ \mathit{cut\text{-}tac}
$$

10.1 Properties of the Empty Set

In Powerset The empty set belongs to the powerset of its generic parameter: $\varnothing[S] \in \mathbb{P} S$. To prove this, first we specialize the instance of the empty set in the consequent, then apply the powerset rule. This gives us $y \in \{x : S \mid \mathit{false}\}$ as an antecedent; by set comprehension and schema construction this simplifies to (*inter alia*) *false* as an antecedent. Thus the goal is proved.

$$\begin{array}{l}\mathit{empty\text{-}in\text{-}power} := \mathit{GenSpec} \,;\, \mathit{subst\text{-}tac} \,;\, \mathit{t\text{-}powerset};\\ \qquad\qquad \mathit{setcomp\text{-}t} \,;\, \mathit{thinr\text{-}tac}\,1 \,;\, \mathit{SConstrDdag} \,;\, \mathit{false\text{-}t}\end{array}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ldots \ddagger y : \{x : s \mid \mathit{false} \bullet x\} \ddagger \exists x : s \mid \mathit{false} \bullet y = x \ddagger x : s \ddagger \mathit{false} \vdash y \in S}}{\ldots \ddagger y : \{x : s \mid \mathit{false} \bullet x\} \ddagger \exists x : s \mid \mathit{false} \bullet y = x \ddagger x : s \mid \mathit{false} \vdash y \in S}}{\ldots \ddagger y : \{x : s \mid \mathit{false} \bullet x\} \ddagger \exists x : s \mid \mathit{false} \bullet y = x \ddagger x : s \mid \mathit{false} \ddagger y = x \vdash y \in S}}{\ldots \ddagger y : \{x : s \mid \mathit{false} \bullet x\} \vdash y \in S}}{\ldots \vdash \forall y : \{x : s \mid \mathit{false} \bullet x\} \bullet y \in S}}{\ldots \vdash \{x : s \mid \mathit{false} \bullet x\} \in \mathbb{P} S}}{\varnothing[X] := \{x : X \mid \mathit{false} \bullet x\} \vdash \{X := s\}_{\ominus} \{x : X \mid \mathit{false} \bullet x\} \in \mathbb{P} S}}{\varnothing[X] := \{x : X \mid \mathit{false} \bullet x\} \vdash \varnothing[s] \in \mathbb{P} S}$$

Empty set has no members: $y \in \varnothing \vdash \mathit{false}$. To prove this, we expand the definition of $\varnothing$ using a cut, using *genspec-assum* to discharge the resulting goal.

$$\begin{array}{l}\mathit{in\text{-}empty\text{-}t} := \mathit{cut\text{-}tac}(y \in \{x : S \mid \mathit{false}\});\\ \qquad (\mathit{cut\text{-}tac}(\varnothing[S] = \{x : S \mid \mathit{false}\});\\ \qquad (\mathit{thinr\text{-}tac}\,1 \,;\, \mathit{genspec\text{-}assum}\\ \qquad\quad \|\mathit{symm\text{-}t} \,;\, \mathit{t\text{-}leibniz} \,;\, \mathit{thinr\text{-}tac}\,1 \,;\, \mathit{assum\text{-}tac})\\ \qquad \|\mathit{setcomp\text{-}t} \,;\, \mathit{thinr\text{-}tac}\,1 \,;\, \mathit{SConstrDdag} \,;\, \mathit{false\text{-}t})\end{array}$$

$$\dfrac{\dfrac{\dfrac{}{\varnothing[X'_1 := \{x : X_1 \mid \mathit{false}\}} \; \mathit{genspec\text{-}assum}}{\varnothing[X'_1 := \{x : X_1 \mid \mathit{false}\} \ddagger y \in \varnothing[S]} \; \mathit{thinr}}{\varnothing[X] := \{x : X \mid \mathit{false}\} \ddagger y \in \varnothing[S] \vdash y \in \{x : S \mid \mathit{false}\}} \; \mathit{cut\text{-}tac}$$

$$\dfrac{\dfrac{}{\ddagger y \in \varnothing[S] \ddagger \{x : S \mid \mathit{false}\} = \varnothing[S] \vdash y \in \varnothing[S]} \; \mathit{iter\text{-}assum\text{-}tac}}{\ddagger y \in \varnothing[S] \ddagger \{x : S \mid \mathit{false}\} = \varnothing[S] \vdash y \in \{x : S \mid \mathit{false}\}} \; \mathit{t\text{-}leibniz}$$

$$\dfrac{\dfrac{\dfrac{}{\ldots \ddagger y : S \ddagger \mathit{false} \vdash \mathit{false}} \; \mathit{false\text{-}t}}{\ddagger y : S \mid \mathit{false} \vdash \mathit{false}} \; \mathit{SConstrDdag}}{\ddagger y : S \mid \mathit{false} \ddagger y = x \vdash \mathit{false}} \; \mathit{thinr}}{\ddagger y \in \{x : S \mid \mathit{false}\} \vdash \mathit{false}} \; \mathit{setcomp\text{-}t}$$

$$\dfrac{\dfrac{\ddagger y \in \varnothing[S] \ddagger \{x : S \mid \mathit{false}\} = \varnothing[S] \vdash y \in \varnothing[S]}{\ddagger y \in \varnothing[S] \ddagger \{x : S \mid \mathit{false}\} = \varnothing[S] \vdash y \in \{x : S \mid \mathit{false}\}} \; \mathit{symm\text{-}t}}{\varnothing[X] := \{x : X \mid \mathit{false}\} \ddagger y \in \varnothing[S] \vdash \mathit{false}} \; \mathit{cut\text{-}tac}$$

This lemma offers an alternative proof of the previous one—expand the powerset first, and then do the generic instantiation in the antecedent. This turns out to take many more steps.

When a specialized version of the generic definition is already available, the proof of this lemma can be considerably simplified.

$$in\text{-}empty\text{-}tt := leibniz\text{-}t;\ setcomp\text{-}t;\ thinr\text{-}tac\,1;\ SConstrDdag;\ false\text{-}t$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{x : S \mid false \bullet x\} = \varnothing[S] \ddagger \exists x : S \mid false \bullet y = x \ddagger x : S \ddagger false \vdash false}\ false\text{-}t}{\{x : S \mid false \bullet x\} = \varnothing[S] \ddagger \exists x : S \mid false \bullet y = x \ddagger x : S \mid false \vdash false}\ SConstrDdag}{\{x : S \mid false \bullet x\} = \varnothing[S] \ddagger \exists x : S \mid false \bullet y = x \ddagger x : S \mid false \ddagger y = x \vdash false}\ thinr}{\{x : S \mid false \bullet x\} = \varnothing[S] \ddagger y \in \{x : S \mid false \bullet x\} \vdash false}\ setcomp\text{-}t}{\varnothing[S] = \{x : S \mid false \bullet x\} \ddagger y \in \varnothing[S] \vdash false}\ leibniz\text{-}t$$

The Empty set is a partial function $\varnothing \in S \pfun T$. The proof of this lemma appeals to some of the simpler lemmas already proven.

$$empty\text{-}in\text{-}pfun := pfun\text{-}is\text{-}rel\text{-}and\text{-}fun;\ (rel\text{-}is\text{-}power;\ t\text{-}bindprod;\ empty\text{-}in\text{-}power \parallel empty\text{-}is\text{-}fun)$$

$$\dfrac{\dfrac{\dfrac{}{\varnothing[X, Y] := \{x : X \mid false\}}\ empty\text{-}in\text{-}power}{\varnothing[X, Y] := \{x : X \mid false\} \vdash \varnothing[S \times T] \in \mathbb{P}(S \times T)}\ t\text{-}bindprod}{\varnothing[X, Y] := \{x : X \mid false\} \vdash \{f := \varnothing[S \times T]\} \in [f : \mathbb{P}(S \times T)]}\ rel\text{-}is\text{-}power \qquad \dfrac{}{\varnothing[X, Y] := \{x : X \mid false\} \ddagger S \pfun T = \mathbb{P}(S \times T) \vdash \forall x_1 : S;\ y_1, y_2 : T \bullet (x_1, y_1) \in \varnothing[S \times T] \land (x_1, y_2) \in \varnothing[S \times T] \Rightarrow y_1 = y_2\}}\ empty\text{-}is\text{-}fun$$

$$\dfrac{\varnothing[X, Y] := \{x : X \mid false\} \ddagger \{f := \varnothing[S \times T]\} \in [f : S \pfun T]}{\varnothing[X, Y] := \{x : X \mid false\} \ddagger S \pfun T = \mathbb{P}(S \times T) \ddagger S \to T = \{f : S \pfun T \mid \forall x_1 : S;\ y_1, y_2 : T \bullet (x_1, y_1) \in f \land (x_1, y_2) \in f \Rightarrow y_1 = y_2\} \vdash \varnothing[S \times T] \in S \to T}\ pfun\text{-}is\text{-}rel\text{-}and\text{-}fun$$

An additional lemma is needed, to show that has the functional property given in the partial function definition.

```
empty-is-fun := cut-tac(∅[S × T] = {x : S × T | false});
                (thinr-tac 1 ; genspec-assum‖
                t-all ; t-imp ; and-t;
                lshift 4 ; lshift 2;
                leibniz-t ; setcomp-t ; thinr-tac 1;
                SConstrDdag ; false-t)
```

The domain of the empty function is the empty set: $\varnothing = \mathrm{dom}\,\varnothing$.

```
empty-dom := lshift 4 ; all-t⌊ R := ∅[NAME × DATE] ⌋;
             (BindProd ; subst-tac ; lshift 4;
             t-leibniz ; thinr-tac 2 ; empty-in-power
             ‖
             subst-tac ; trans-tac ; t-seteq;
             (drop-snd ; drop-snd ; drop-snd;
             dec-in-t ; in-empty-t-NAME
             ‖setcomp-t;
             thinr-tac 1 ; SConstrDdag;
             lshift 3 ; thinr-tac 1 ; falseE;
             ↷2 thinr-tac 4 ; in-empty-t))
```

Discovering how empty-dom works, and improving on its structure, is left as an exercise for the reader.

Part III

Schemas, and Proofs about Specifications

For the purposes of constructing proofs about 'real' Z specifications, we must be able to use the schema calculus and to be able to expand schema definitions. This means that we must have ways of considering schemas as expressions and as predicates, as well as declarations.

The schema calculus has evolved from a particular view of the semantics of schemas. Rather than considering the view of schemas as expressions, having a type and a value as a set of bindings, these operations have been defined using a view based on signatures and properties. Such a view is somewhere in between an expression and a predicate.

This final part of the tutorial gives a comprehensive definition of schemas and uses these rules in constructing the proofs of the two theorems about the birthday book set out in part one.

Contents

11 Schema Declarations

Schemas are used as declarations in many places within a specification. Some of these occurrences are not immediately obvious. So we begin by looking at the three ways in which a schema as a declaration can arise.

The purpose of a declaration is to introduce new names and to give them values. So from a schema declaration we must be able to extract its property. This extraction process allows us to reason about the values of the variables. However due to the encapsulation of names by schemas, great care must be taken when calculating the side conditions.

11.1 Instances of Schema Declarations

A schema can be used to introduce its component names whose values satisfy its property. When used in a sequent a schema declaration appears in the antecedent as follows:

$$\cdots \; ‡S‡ \cdots \vdash \cdots$$

This feature of schemas is used in three different ways: as a schema-text, a schema-inclusion, or as an axiomatic definition.

Axiomatic Definition An axiomatic definition is a particular piece of syntax used to introduce new names into a specification. For example the following definition

$$\begin{array}{|l} x, y : \mathbb{Z} \\ \hline x \neq y \end{array}$$

introduces two numbers x and y and states that they are not equal. The schema $x, y : \mathbb{Z} \mid x \neq y$ contains the same information and could just as well have been used to introduce the two names:

$$x, y : \mathbb{Z} \mid x \neq y$$

So, for any schema we can introduce its component names together with its property simply by stating it as a declaration. In the notation used in the antecedent of a sequent we would write the following:

$$\cdots ‡x, y : \mathbb{Z} \mid x \neq y‡ \cdots \vdash \cdots$$

Schema Inclusion The typical example of a schema inclusion is found in definitions of more complex schemas. In the birthday book example, the schema *AddBirthday* is defined in terms of Δ*BirthdayBook* and other variables:

AddBirthday

$\Delta BirthdayBook$;
$name? : NAME$;
$date? : DATE$

$\neg\ (name? \in known) \land$
$birthday' = birthday \cup \{(name?, date?)\}$

This paragraph comprises a definition of a schema name *AddBirthday* using a schema constructed from an inclusion. We would write it in the sequent notation as follows:

$$AddBirthday := \left[\begin{array}{l} \Delta BirthdayBook; \\ name? : NAME; \\ date? : DATE \end{array} \;\middle|\; \begin{array}{l} \neg\ (name? \in known) \land \\ birthday' = birthday \cup \{(name?, date?)\} \end{array}\right] \ .$$

This is a schema composition of the form $[S \mid P]$ where the schema S is included in the schema, together with the predicate P.

The rule *SchConstrPar* allows us to split a schema-construction into its schema inclusion and predicate making them into a schema declaration and a constraint paragraph:

$$\frac{S \natural P \vdash Q}{S \mid P \vdash Q}\ SchConstrPar$$

Schema Text A schema text is used in quantified predicates such as $\forall S \bullet P$. The rule *AllI*

$$\frac{\Gamma \natural S \vdash P}{\Gamma \vdash \forall S \bullet P}\ \uparrow\downarrow AllI$$

decomposes a universal quantification by generating a schema declaration and a simpler predicate.

The rules for set comprehension and definite description give us quantified predicates that are also proved using *AllI*. Thus this is a very common route for introducing schema declarations into the antecedent of a sequent.

Bindings The generalisation of the meaning of paragraphs to encompass arbitrary schemas can be repeated for substitutions. A substitution $\{\ x := e\ \}$ is a particular form of a binding. We can use the same techniques for taking a schema and making it a declaration and give a substitution semantics for any binding or appropriately typed expression.

11.2 Properties of Schema Declarations

Given a schema declaration in the antecedent of a sequent, we can also assume its property. We use square brackets to indicate that we are interpreting it as a schema predicate:

$$
\begin{aligned}
\alpha(S \mid P) &= \alpha S \\
\\
\alpha \neg S &= \alpha S \\
\alpha S \diamond T &= \alpha S \cup \alpha T \\
\alpha \forall S \bullet T &= \alpha T \setminus \alpha S \\
\alpha \exists S \bullet T &= \alpha T \setminus \alpha S
\end{aligned}
$$

Figure 4: The Alphabets of Schemas

$$
\frac{\Gamma \ddagger S \ddagger [S] \vdash P}{\Gamma \ddagger S \vdash P}\ SchProp(\mathbf{wf}\, S)
$$

Since the alphabet of a schema S is dependent on its definition, we cannot use the scoping variable technique in this case.

In order to take sequents of this form any further, we must look at schemas as predicates.

11.3 Alphabet

The alphabet of a schema declaration is dependent on the context in which it is calculated. The alphabet schema reference can be calculated only when the signature of the schema has been discovered. The discovery process follows the definition of the schema reference using the following rule:

$$
\overline{\Gamma \ddagger S := T \vdash \alpha S = \alpha T}
$$

The alphabets of composite schemas are defined in terms of the alphabets of their sub-schemas. The rules for this calculation are given in figure 4.

11.4 Free Variables and Substitution

The free variables of a schema declaration are the same as the free variables of the schema as an expression.

The substitution rules for schema declarations are the same as for schema expressions.

12 Schema Predicates

Schema predicates are just a special form of predicate. When treated as predicates, schemas behave in exactly the same way as ordinary predicates. It is, however, important to distinguish the two forms of connectives. Though they

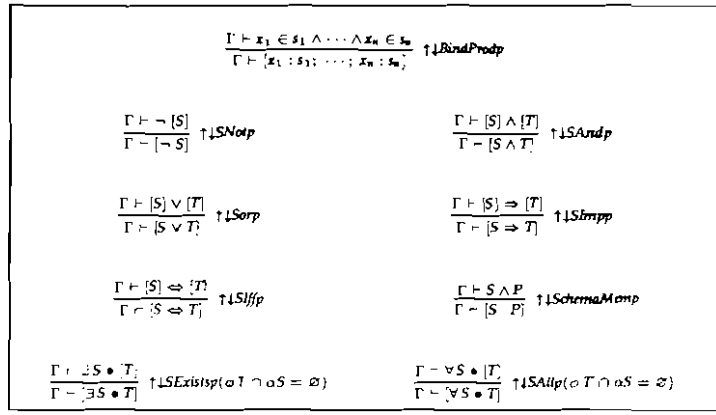


Figure 5: The Schema Predicate Calculus

look the same the propositional and schema connectives operate in subtly different ways. For all well formed instances of schemas, there is no difference.

We first look at what it means to view a schema as a predicate, and then look at the laws governing the schema connectives.

12.1 Interpretation

The simple schema

$$x : s$$

can be said to be true whenever the variable x has a value which is a member of the the set s. This condition can be expressed as follows:

$$\frac{\Gamma \vdash x \in s}{\Gamma \vdash [x : s]} \uparrow\downarrow(\mathbf{wf}\, x : s)$$

This more general case of the schema S raises some problems. What does it mean to say that S is true?

12.2 Definition of Schema Connectives

When we consider the schema operators corresponding to predicates, they have the same properties as ordinary predicates. The schema construction $S \mid P$ can (when viewed as a predicate) be considered to be a conjunction of a schema predicate and an ordinary predicate. The rules illustrating these properties are given in figure 5.

Simplifying Schema Predicates Just like the for the predicate calculus, we can develop a tactic for simplifying schema predicate formulae. Since we will want it to work for schema predicates in both the antecedent and the consequent we make use of the reversible rules and generate two-sided tactics:

$$sand\text{-}p := up\text{-}down\text{-}tacSAndp$$

$$bindprod\text{-}p := up\text{-}down\text{-}tacBindProdp$$

$$smem\text{-}p := up\text{-}down\text{-}tacSchemaMemp$$

We can examine how these rules are used to simplify the property of the expanded *Birthday* schema:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{k \in \mathbb{P} N \ \ddagger b \in N \pfun D \ \ddagger k = \mathrm{dom}\, b \ \vdash \ P}{k \in \mathbb{P} N \ \ddagger [b : N \pfun D] \ \ddagger k = \mathrm{dom}\, b \ \vdash \ P}\ \curvearrowleft bindprod\text{-}p}{[k : \mathbb{P} N] \ \ddagger [b : N \pfun D] \ \ddagger k = \mathrm{dom}\, b \ \vdash \ P}\ \curvearrowright bindprod\text{-}p}{[k : \mathbb{P} N] \wedge [b : N \pfun D] \ \ddagger k = \mathrm{dom}\, b \ \vdash \ P}\ \curvearrowleft and\text{-}t}{[k : \mathbb{P} N;\ b : N \pfun D] \ \ddagger k = \mathrm{dom}\, b \ \vdash \ P}\ \curvearrowright s\text{-}and\text{-}p}{[k : \mathbb{P} N;\ b : N \pfun D] \wedge k = \mathrm{dom}\, b \ \vdash \ P}\ and\text{-}t}{[k : \mathbb{P} N;\ b : N \pfun D \mid k = \mathrm{dom}\, b] \ \vdash \ P}\ smem\text{-}p$$

The tactic *sch-pred-t*[10] follows the pattern in this simplification by splitting the schema predicate into a conjunction of schema predicates, and then separates them and repeats the process on both of the sub-expressions:

$$\begin{aligned} sch\text{-}pred\text{-}t := \ & !((bindprod\text{-}p \mid smem\text{-}p \mid sand\text{-}p \mid \mathbf{skip}); \\ & (and\text{-}t;\ \curvearrowleft sch\text{-}pred\text{-}t;\ sch\text{-}pred\text{-}t \mid \mathbf{skip}) \\ & \mid \mathbf{skip}) \end{aligned}$$

12.3 Schema Predicate Substitution

The free variables of a schema as a predicate are generated from two sources. The first is the free variables of the schema as an expression. The second source is the alphabet of the schema. These component names are newly introduced by the schema. So we have the following equality:

$$\Phi S = \phi S \cup \alpha S$$

The additional free variables in a schema predicate result in a different substitution. Not only can the expression level variables be substituted, but also the component variables. Sometimes they can be the same names. This can cause some complications for the quantified schemas. The substitution rules for schema predicates are given in figure 6.

[10] We demonstrate a simplified tactic that only considers three types of constructor: those that generate predicate conjunctions.

$$
\begin{array}{rcl}
b \odot [S \mid P] & \equiv & b \odot S \land b \odot P \\
\\
b \odot [\neg S] & \equiv & \neg b \odot S \\
b \odot [S \circ T] & \equiv & b \odot S \circ b \odot T \\
b \odot [\forall S \bullet T] & \equiv & \forall b \ominus S \bullet b \odot T \\
 & & \alpha(b \ominus S) \cap (\phi b \ominus T \cup \alpha b) = \varnothing \\
\\
b \odot [\exists S \bullet T] & \equiv & \exists b \ominus S \bullet b \odot T \\
 & & \alpha(b \ominus S) \cap (\phi b \ominus T \cup \alpha b) = \varnothing
\end{array}
$$

Figure 6: Substitution into Schema Predicates

13 Schema Expressions

A more complete definition of schemas is obtained from looking at them as expressions. These schema expressions are sets of bindings. We characterise these sets by defining the property of membership. So each rule considers what it means for a binding to be in a particular schema construction. The rules for schema expressions are given in figure 7.

Schema Expression Substitution The difference in free variables for schemas as expressions and as predicates means that there are two types of substitution that must be considered. Substitution into schema expressions is a homomorphism. The rules for schema expression substitution are given in figure 8.

14 Specification-Level Proofs

We can now revisit the proofs presented at the beginning of the tutorial.

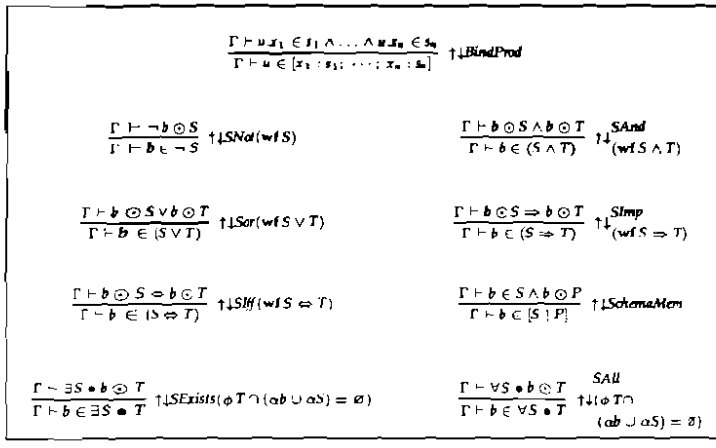


Figure 7: The Schema Expression Calculus

$$
\begin{array}{rcl}
b \ominus [S \mid P] & \equiv & [b \ominus S \mid b \odot P] \\
 & & \textbf{when } \alpha b \cap \alpha S = \varnothing \\
b \ominus [S \mid P] & \equiv & [b \ominus S \mid P] \\
 & & \textbf{when } \alpha b \cap \Phi P \subseteq \alpha S \\
 & & \\
b \ominus [\neg S] & \equiv & [\neg b \ominus S] \\
 & & \\
b \ominus [S \diamond T] & \equiv & [b \ominus S \diamond b \ominus T] \\
b \ominus [\forall S \bullet T] & \equiv & [\forall b \ominus S \bullet b \ominus T] \\
b \ominus [\exists S \bullet T] & \equiv & [\exists b \ominus S \bullet b \ominus T]
\end{array}
$$

Figure 8: Substitution into Schema Expressions

14.1 Theorem about *AddBirthday*

Section 6.3 demonstrated how the proof given by Spivey can be set up as a tactic over the inference rules presented here.

$$
\begin{aligned}
&\mathit{spivey}\text{-}5 := \mathit{eqtac}(\mathit{lemma}1)\\
&\qquad\qquad (\mathit{known}' = \mathrm{dom}\,\mathit{birthday}');\\
&\qquad \mathit{eqtac}(\mathit{lemma}2)\\
&\qquad\qquad (\mathit{birthday}' = \mathit{birthday} \cup \{(\mathit{name}?, \mathit{date}?)\});\\
&\qquad \mathit{eqtac}(\mathit{lemma}3)\\
&\qquad\qquad (\mathrm{dom}(\mathit{birthday} \cup \{(\mathit{name}?, \mathit{date}?)\}) =\\
&\qquad\qquad\qquad\qquad (\mathrm{dom}\,\mathit{birthday}) \cup (\mathrm{dom}\{(\mathit{name}?, \mathit{date}?)\}));\\
&\qquad \mathit{eqtac}(\mathit{lemma}4)\\
&\qquad\qquad (\mathrm{dom}\{(\mathit{name}?, \mathit{date}?)\} = \{\mathit{name}?\});\\
&\qquad \mathit{eqtac}(\mathit{lemma}5)\\
&\qquad\qquad (\mathrm{dom}\,\mathit{birthday} = \mathit{known});\\
&\qquad \mathit{Refl}
\end{aligned}
$$

In order to complete the proof, we must simply demonstrate that the five lemmas used above follow from the definition of *AddBirthday*.

$$\mathit{AddBirthday} \vdash \mathit{known}' = \mathrm{dom}\,\mathit{birthday}'$$

$$\mathit{AddBirthday} \vdash \mathit{birthday}' = \mathit{birthday} \cup \{(\mathit{name}?, \mathit{date}?)\}$$

$$\mathit{AddBirthday} \vdash \mathrm{dom}(\mathit{birthday} \cup \{(\mathit{name}?, \mathit{date}?)\}) = (\mathrm{dom}\,\mathit{birthday}) \cup (\mathrm{dom}\{(\mathit{name}?, \mathit{date}?)\})$$

$$\mathit{AddBirthday} \vdash \mathrm{dom}\{(\mathit{name}?, \mathit{date}?)\} = \{\mathit{name}?\}$$

$$\mathit{AddBirthday} \vdash \mathrm{dom}\,\mathit{birthday} = \mathit{known}$$

Several of the lemmas arise directly from the definitions in the antecedent. The tactic *gen-expand* expands schema definitions as far as possible. It works on goals such as

$$
\begin{aligned}
&S_1 := T\ddagger\\
&S_2 := [S_1 \mid P]\ddagger\\
&S_3 := [S_2;\ S_2']\ddagger\\
&S_3 \vdash Q
\end{aligned}
$$

in which S_3 is rewritten to give

$$T\ddagger P\ddagger T'\ddagger P_1 \vdash Q$$

etc (Here P_1 is the predicate which results from replacing all those variables in P which belong to the alphabet of T by their primed versions.)

First, the schema expressions are expanded, according to their definitions; then any resulting decorated schemas are rewritten so that only variable names

are decorated; a predicate copy of the schema is made, and then it is simplified into ‡-separated components.

$$gen\text{-}expand := !(exhaust\ leibniz\text{-}t;\ try\ decorate\text{-}left);\ par\text{-}pred\text{-}t;\ sch\text{-}pred\text{-}t$$

When *gen-expand* is applied to our initial goal, or one of the lemmas which remains to be proved, the result is as follows:

$$\frac{\begin{array}{l} known : \mathbb{P}\, NAME \land birthday : NAME \pfun DATE \mid known = \mathrm{dom}\, birthday \land \\ \quad known' : \mathbb{P}\, NAME \land birthday' : NAME \pfun DATE \mid known' = \mathrm{dom}\, birthday' \land \\ \quad name? : NAME \land date? : DATE \mid \\ \quad \neg\ (name? \in known) \land birthday' = birthday \cup \{(name?, date?)\} \\ known \in \mathbb{P}\, NAME \\ birthday \in NAME \pfun DATE \\ known = \mathrm{dom}\, birthday \\ known' \in \mathbb{P}\, NAME \\ birthday' \in NAME \pfun DATE \\ known' = \mathrm{dom}\, birthday' \\ name? \in NAME \\ date? \in DATE \\ \neg\ (name? \in known) \\ birthday' = birthday \cup \{(name?, date?)\} \\ \vdash \\ Q \end{array}}{\begin{array}{l} BirthdayBook := [known : \mathbb{P}\, NAME;\ birthday : NAME \pfun DATE \mid known = \mathrm{dom}\, birthday] \\ \Delta BirthdayBook := [BirthdayBook,\ BirthdayBook']' \\ AddBirthday := [\Delta BirthdayBook;\ name? : NAME;\ date? : DATE \mid \\ \qquad \neg\ (name? \in known) \land birthday' = birthday \cup \{(name?, date?)\}] \\ AddBirthday \\ \vdash \\ Q \end{array}}\ gen\text{-}expand$$

Having done this, in order to discharge a goal which appears somewhere in the schema antecedent, we need simply the tactic *is-known*:

$$is\text{-}known := gen\text{-}expand;\ iter\text{-}assum\text{-}tac$$

Simple Lemmas Therefore, the tactics to prove some of the lemmas are now entirely trivial.

$$lemma\,1 := is\text{-}known$$

$$lemma\,2 := is\text{-}known$$

$$lemma\,5 := is\text{-}known$$

Lemmas that use the toolkit The remaining two lemmas rely on properties of the toolkit definitions, some of which were proved earlier. Since a common

form for these lemmas arising from the toolkit is $\vdash \forall S \bullet P$, we provide a tactical *toolkit*, which takes three arguments—the tactic used to prove the lemma, the predicate which is the lemma, and a binding showing how the quantification is to be specialized in this case—and returns (assuming the tactic succeeds) the requirement to show that the binding belongs to the schema in the quantifier (i.e. that it has the right type).

This last requirement is generally covered by the tactic *is-known-type*, which takes this resulting goal, simplifies it, and applies *is-known* where possible.

$$\begin{aligned} &\mathit{is\text{-}known\text{-}type} := (\mathit{Sand} \mid \mathit{BindProd} \mid \mathit{SchemaMem}); \\ &\qquad\qquad \mathit{t\text{-}and} \,;\, !(\mathit{subst\text{-}tac};\ \mathit{is\text{-}known} \mid \mathit{subst\text{-}tac}) \end{aligned}$$

Lemma 4 The fourth lemma uses the so-called 'Harwood's Theorem':

$$\forall a : A;\ b : B \bullet \mathrm{dom}\{(a,b)\} = \{a\}$$

This has been proved elsewhere, by *harwood-tac*.

$$\begin{aligned} &\mathit{lemma4} := \mathit{toolkit}(\mathit{harwood\text{-}tac}) \\ &\qquad\qquad (\forall a : \mathit{NAME};\ b : \mathit{DATE} \bullet \mathrm{dom}\{(a,b)\} = \{a\}) \\ &\qquad\qquad \{\, a := \mathit{name?}, b := \mathit{date?}\,\}; \\ &\qquad \mathit{is\text{-}known\text{-}type} \end{aligned}$$

$$\frac{\overline{\mathit{AddBirthday} \vdash \{\, a := \mathit{name?}, b := \mathit{date?}\,\} \in [a : \mathit{NAME};\ b : \mathit{DATE}]}\ \mathit{is\text{-}known\text{-}type}}{\mathit{AddBirthday} \vdash \mathrm{dom}\{(\mathit{name?}, \mathit{date?})\} = \{\mathit{name?}\}}\ \mathit{toolkit}(\mathit{harwood\text{-}tac})\,.$$

Lemma 3 The third lemma depends on the 'toolkit' lemma

$$\forall f, g : \mathit{NAME} \pfun \mathit{DATE} \bullet \mathrm{dom}(f \cup g) = \mathrm{dom}\, f \cup \mathrm{dom}\, g$$

which will be proved by *dom-cup-lemma* — omitted. In this instance, the *toolkit* tactic produces the subgoal

$$\begin{aligned} &\mathit{AddBirthday} \\ &\vdash \\ &\{\, f := \mathit{birthday}, g := \{(\mathit{name?}, \mathit{date?})\}\,\} \in [f, g : \mathit{NAME} \pfun \mathit{DATE}] \end{aligned}$$

which simplifies—via the first part of *is-known-type*—to

$$\mathit{AddBirthday} \vdash \mathit{birthday} \in \mathit{NAME} \pfun \mathit{DATE}$$

$$\mathit{AddBirthday} \vdash \{(\mathit{name?}, \mathit{date?}\} \in \mathit{NAME} \pfun \mathit{DATE}$$

The first of these is discharged immediately by *is-known*; the second requires more work—making use of the tactics relating partial functions, relations, singleton pairs, powersets, etc., in Section 9, above.

```
use-domcup := dom-cup-lemma

lemma3 := toolkit(use-domcup)
                (∀f, g : NAME ⇸ DATE • dom(f ∪ g) =
                        dom f ∪ dom g)
                (f := birthday, g := {(name?, date?)} );
          is-known-type; nd-fun

nd-fun := lshift 7; lshift 8; pfun-is-rel-and-fun;
          (rel-is-power; pair-pow-prod-mem;
          (thinr; is-known)
          ||
          !(t-all; t-imp; and-t; extmem-t; tuplesel-t;
          thinr-tac 2; extmem-t; tuplesel-t;
          Refl))
```

Of course, this proof has been arranged for readability. It is very far from being an efficient proof—expanding *AddBirthday* separately for each lemma is very costly. Conversely, part of the power of Z is in the ability it gives the user to wrap up information in a schema—and experience shows that with proofs involving sizeable specifications it is most important not to expand schema definitions fully until the information they contain is needed.

14.2 Initialization Theorem

The tactic that follows is perhaps a more honest 'first-cut' proof of the initialization theorem. It begins by providing a witness for the existential quantifier—the empty function *birthday* and empty set *known*. Using this binding, it is easy to discharge the original goal; the new goal is to prove that this binding does indeed belong to *InitBirthdayBook*.

This is done by replacing *InitBirthdayBook* by its definition (using *Leibniz*). The predicate part of *InitBirthdayBook* is then quickly solved by *Refl*. The remaining goal is further expanded by use of the definition of *Birthday*—which was included by *InitBirthdayBook*. After making all the resulting substitutions, and applying *t-and* several times, we are left to demonstrate three properties of the empty set; these were proved in Section 9.

$$
\begin{aligned}
&\mathit{init\text{-}tac} := \mathit{t\text{-}exists}(\,\mathit{known} := \varnothing[\mathit{NAME}], \mathit{birthday} := \varnothing[\mathit{NAME} \times \mathit{DATE}]\,);\\
&\qquad (\mathit{subst\text{-}tac}\ ;\ \mathit{t\text{-}true} \parallel \mathbf{skip});\\
&\qquad \mathit{t\text{-}leibniz}\ ;\ \mathit{SchBindMem};\\
&\qquad \mathit{subst\text{-}tac}\ ;\ \mathit{t\text{-}and};\\
&\qquad (\mathbf{skip} \parallel \mathit{Refl})\\
&\qquad \mathit{thinr\text{-}tac2}\ ;\ \mathit{t\text{-}leibniz};\\
&\qquad \mathit{thinr\text{-}tac2}\ ;\ \mathit{t\text{-}and};\\
&\qquad (\mathit{t\text{-}and} \parallel \mathbf{skip});\\
&\qquad ((\mathit{empty\text{-}in\text{-}power} \parallel \mathit{empty\text{-}in\text{-}pfun}) \parallel \mathit{empty\text{-}dom})
\end{aligned}
$$

Ignoring the minor subgoals, then, the tactic above produces a proof with this outline form:

$$
\dfrac{\dfrac{\dfrac{}{\varnothing[\mathit{NAME}] \in \mathbb{P}\,\mathit{NAME}}\ \mathit{e\text{-}in\text{-}pow} \quad \dfrac{}{\vdash \varnothing[\mathit{NAME} \times \mathit{DATE}] \in \mathit{NAME} \to \mathit{DATE}}\ \mathit{e\text{-}in\text{-}pfun} \quad \dfrac{}{\vdash \varnothing[\mathit{NAME}] = \operatorname{dom} \varnothing[\mathit{NAME} \times \mathit{DATE}]}\ \mathit{e\text{-}dom}}{\vdash}\ \mathit{t\text{-}and}
$$

$$
\dfrac{\begin{array}{l}\mathit{bindknown} := \varnothing[\mathit{NAME}], \mathit{birthday} := \varnothing[\mathit{NAME} \times \mathit{DATE}] \odot\\ \quad [\mathit{known} : \mathbb{P}\,\mathit{NAME};\ \mathit{birthday} : \mathit{NAME} \to \mathit{DATE} \mid \mathit{known} = \operatorname{dom} \mathit{birthday}]\end{array}}{\begin{array}{l}\mathit{BirthdayBook} := [\mathit{known} : \mathbb{P}\,\mathit{NAME};\ \mathit{birthday} : \mathit{NAME} \to \mathit{DATE} \mid \mathit{known} = \operatorname{dom} \mathit{birthday}]\\ \vdash\\ (\,\mathit{known} := \varnothing[\mathit{NAME}], \mathit{birthday} := \varnothing[\mathit{NAME} \times \mathit{DATE}] \odot \mathit{BirthdayBook} \wedge\\ \varnothing[\mathit{NAME}] = \varnothing[\mathit{NAME}]\end{array}}\ \mathit{t\text{-}leibniz}
$$

$$
\dfrac{\begin{array}{l}\mathit{BirthdayBook} := [\mathit{known} : \mathbb{P}\,\mathit{NAME};\ \mathit{birthday} : \mathit{NAME} \to \mathit{DATE} \mid \mathit{known} = \operatorname{dom} \mathit{birthday}];\\ \mathit{InitBirthdayBook} := [\mathit{BirthdayBook} \mid \mathit{known} = \varnothing[\mathit{NAME}]]\\ \vdash\\ (\,\mathit{known} := \varnothing[\mathit{NAME}], \mathit{birthday} := \varnothing[\mathit{NAME} \times \mathit{DATE}] \in \mathit{InitBirthdayBook}\end{array}}{\begin{array}{l}\mathit{BirthdayBook} := [\mathit{known} : \mathbb{P}\,\mathit{NAME};\ \mathit{birthday} : \mathit{NAME} \to \mathit{DATE} \mid \mathit{known} = \operatorname{dom} \mathit{birthday}];\\ \mathit{InitBirthdayBook} := [\mathit{BirthdayBook} \mid \mathit{known} = \varnothing[\mathit{NAME}]]\\ \vdash\\ \exists\, \mathit{InitBirthdayBook} \bullet \mathit{true}\end{array}}\ \mathit{t\text{-}exists}
$$

References

Harwood, W. T. (1990). Proof rules for Balzac, *Technical report*, ISTL Cambridge.

Jones, R. B. (1990). Proof support for Z via HOL parts I & II, *Technical report*, ICL Secure Systems, Winnersh.

Martin, A. P., Gardiner, P. H. B. and Woodcock, J. C. P. (1996). A tactic calculus, *Formal Aspects of Computing*. To appear.

Sennett, C. T. (1987). Review of type checking and scope rules of the specification language Z, *Report no. 87017*, RSRE, Ministry of Defence, Malvern, Worcestershire, UK.

Spivey, J. M. (1992). *The Z Notation: A Reference Manual*, second edn, Prentice-Hall.

Woodcock, J. C. P. and Brien, S. M. (1992). *W*: A logic for Z, *in* J. E. Nicholls (ed.), *Proceedings of the Sixth Annual Z User Meeting, 1991*, Workshops in Computing, Springer-Verlag, pp. 77–96.

Appendix

This appendix records the text of the relevant parts of the current draft Z standard. This is by no means in its final form, but represents the best available so far.

Contents

A Deductive System from Draft Z Standard

A.1 Free variables and alphabets

A.1.1 Paragraphs

$$
\begin{aligned}
\phi[x] &= \varnothing \\
\phi P &= \Phi P \\
\phi S &= \text{See below} \\
\phi(x : s) &= \phi s \\
\phi(x := e) &= \phi e \\
\phi([x]S) &= \phi S \setminus \{x\} \\
\phi(x[y] := e) &= \phi e \\
\phi(\mathbf{\Pi}_1 \ddagger \mathbf{\Pi}_2) &= \phi\mathbf{\Pi}_1 \cup (\phi\mathbf{\Pi}_2 \setminus \alpha\mathbf{\Pi}_1)
\end{aligned}
$$

$$
\begin{aligned}
\alpha[x] &= \{x\} \\
\alpha P &= \varnothing \\
\alpha S &= \text{See below} \\
\alpha(x : s) &= \{x\} \\
\alpha(x := e) &= \{x\} \\
\alpha([x]S) &= \alpha S \\
\alpha(x[y] := e) &= \{x\} \\
\alpha(\mathbf{\Pi}_1 \ddagger \mathbf{\Pi}_2) &= \alpha\mathbf{\Pi}_1 \cup \alpha\mathbf{\Pi}_2
\end{aligned}
$$

A.1.2 Predicates

$$
\begin{aligned}
\Phi(e \in s) &= \phi e \cup \phi s \\
\Phi(e \simeq v) &= \phi e \cup \phi v \\
\Phi \mathbf{true} &= \varnothing \\
\Phi \mathbf{false} &= \varnothing \\
\Phi(\neg P) &= \Phi P \\
\Phi(P \wedge Q) &= \Phi P \cup \Phi Q \\
\Phi(P \vee Q) &= \Phi P \cup \Phi Q \\
\Phi(P \Rightarrow Q) &= \Phi P \cup \Phi Q \\
\Phi(P \Leftrightarrow Q) &= \Phi P \cup \Phi Q \\
\Phi \forall S \bullet P &= \phi S \cup (\Phi P \setminus \alpha S) \\
\Phi \exists S \bullet P &= \phi S \cup (\Phi P \setminus \alpha S) \\
\Phi \exists_1 S \bullet P &= \phi S \cup (\Phi P \setminus \alpha S) \\
\Phi(S) &= \alpha S \cup \phi S \\
\Phi(b \odot P) &= \phi b \cup (\Phi P \setminus \alpha b)
\end{aligned}
$$

A.1.3 Schemas

$$
\begin{aligned}
\phi[x_1 : s_1; \cdots; x_n : s_n] &= \phi(s_1) \cup \cdots \cup \phi(s_n) \\
\phi[S \mid P] &= \phi S \cup (\Phi P \setminus \alpha S) \\
\phi(\neg S) &= \alpha S \\
\phi(S \wedge T) &= \phi S \cup \phi T \\
\phi(S \vee T) &= \phi S \cup \phi T \\
\phi(S \Rightarrow T) &= \phi S \cup \phi T \\
\phi(S \Leftrightarrow T) &= \phi S \cup \phi T \\
\phi(S \upharpoonright T) &= \phi S \cup \phi T \\
\phi(S \backslash [x_1, \ldots, x_n]) &= \phi S \\
\phi(\forall S \bullet T) &= \phi S \cup \phi T \\
\phi(\exists S \bullet T) &= \phi S \cup \phi T \\
\phi(\exists_1 S \bullet T) &= \phi S \cup \phi T \\
\phi(S[x_1/y_1, \ldots, x_n/y_n]) &= \phi S \\
\phi(S \fatsemi T) &= \phi S \cup \phi T \\
\phi(S^q) &= \phi S
\end{aligned}
$$

$$
\begin{aligned}
\alpha[x_1 : s_1, \ldots, x_n : s_n] &= \{x_1, \ldots, x_n\} \\
\alpha[S \mid P] &= \alpha S \\
\alpha(\neg S) &= \alpha S \\
\alpha(S \wedge T) &= \alpha S \cup \alpha T \\
\alpha(S \vee T) &= \alpha S \cup \alpha T \\
\alpha(S \Rightarrow T) &= \alpha S \cup \alpha T \\
\alpha(S \Leftrightarrow T) &= \alpha S \cup \alpha T \\
\alpha(S \upharpoonright T) &= \alpha S \cap \alpha T \\
\alpha(S \backslash [x_1, \ldots, x_n]) &= \alpha S \setminus \{x_1, \ldots, x_n\} \\
\alpha(\forall S \bullet T) &= \alpha T \setminus \alpha S \\
\alpha(\exists S \bullet T) &= \alpha T \setminus \alpha S \\
\alpha(\exists_1 S \bullet T) &= \alpha T \setminus \alpha S \\
\alpha(S[x_1/y_1, \ldots, x_n/y_n]) &= (\alpha S)[x_1/y_1, \ldots, x_n/y_n] \\
\alpha(S \fatsemi T) &= \\
\alpha(S^q) &= (\alpha S)^q \\
\alpha\{\, x_1 := e_1, \ldots, x_n := e_n \,\} &= \{x_1, \ldots, x_n\}
\end{aligned}
$$

A.1.4 Expressions

$$
\begin{aligned}
\phi(x) &= \{x\} \\
\phi(x[e]) &= \{x\} \cup \phi e \\
\phi(i) &= \varnothing \\
\phi(z) &= \varnothing \\
\phi\{e_1, \ldots, e_n\} &= \phi(e_1) \cup \cdots \cup \phi(e_n) \\
\phi\{S \bullet e\} &= \phi S \cup (\phi e \setminus \alpha S) \\
\phi(\mathbb{P}\, s) &= \phi(s) \\
\phi(e_1, \ldots, e_n) &= \phi(e_1) \cup \cdots \cup \phi(e_n) \\
\phi(s_1 \times \cdots \times s_n) &= \phi(s_1) \cup \cdots \cup \phi(s_n) \\
\phi(e.i) &= \phi(e) \\
\phi\{x_1 := e_1, \ldots, x_n := e_n\} &= \phi(e_1) \cup \cdots \cup \phi(e_n) \\
\phi(\theta S) &= \Phi S \\
\phi(b\, x) &= \phi(b) \\
\phi(f e) &= \phi f \cup \phi e \\
\phi(\mu S \bullet e) &= \alpha S \cup \Phi(e_1 \setminus \alpha S) \\
\phi(\textbf{if } P \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) &= \Phi P \cup \phi e_1 \cup \phi e_2 \\
\phi(b \odot e) &= \phi b \cup (\phi e \setminus \alpha b)
\end{aligned}
$$

A.2 Substitution

A.2.1 Predicates

$$
\begin{aligned}
b \odot (e = u) &\equiv b_{\odot} e = b_{\odot} u \\
b \odot (e \in s) &\equiv b_{\odot} e \in b_{\odot} s \\
b \odot \mathbf{true} &\equiv \mathbf{true} \\
b \odot \mathbf{false} &\equiv \mathbf{false} \\
b \odot \neg P &\equiv \neg\, b \odot P \\
b \odot (P \wedge Q) &\equiv b \odot P \wedge b \odot Q \\
b \odot (P \vee Q) &\equiv b \odot P \vee b \odot Q \\
b \odot (P \Rightarrow Q) &\equiv b \odot P \Rightarrow b \odot Q \\
b \odot (P \Leftrightarrow Q) &\equiv b \odot P \Leftrightarrow b \odot Q
\end{aligned}
$$

When $\alpha b \cap \Phi P \subseteq \alpha S$:

$$
\begin{aligned}
b \odot \forall S \bullet P &\equiv \forall b \odot S \bullet P \\
b \odot \exists S \bullet P &\equiv \exists b \odot S \bullet P \\
b \odot \exists_1 S \bullet P &\equiv \exists_1 b \odot S \bullet P
\end{aligned}
$$

When $\alpha b \cap \alpha S \cap \Phi P = \varnothing$ and $\alpha S \cap \phi b = \varnothing$:

$$
\begin{aligned}
b \odot \forall S \bullet P &\equiv \forall b \odot S \bullet b \odot P \\
b \odot \exists S \bullet P &\equiv \exists b \odot S \bullet b \odot P \\
b \odot \exists_1 S \bullet P &\equiv \exists_1 b \odot S \bullet b \odot P
\end{aligned}
$$

When $\alpha b \cap \alpha S = \varnothing$:

$$
b \odot S \equiv [b_{\odot} S]
$$

When $\mathbf{wf}\, b$:

$$
b \odot S \equiv b \odot [b_{\odot} S]
$$

$$
\begin{aligned}
\{ y := v \} \odot (\{ y := u \} \odot P) &\equiv \{ y := \{ y := v \} \odot u \} \odot P \\
\{ x := v \} \odot (\{ y := u \} \odot P) &\equiv \{ y := \{ x := v \} \odot u \} \odot (\{ x := v \} \odot P) \\
&\quad \text{where } y \notin \phi v
\end{aligned}
$$

A.2.2 Schema predicates

$$
\begin{aligned}
b \odot [x_1, \ldots, x_n : s] &\equiv b_{\odot} x_1 \in b_{\odot} s \wedge \ldots \wedge b_{\odot} x_n \in b_{\odot} s \\
b \odot [S \mid P] &\equiv b \odot S \wedge b \odot P \\
b \odot [\neg S] &\equiv \neg b \odot S \\
b \odot [S \wedge T] &\equiv b \odot S \wedge b \odot T \\
b \odot [S \vee T] &\equiv b \odot S \vee b \odot T \\
b \odot [S \Rightarrow T] &\equiv b \odot S \Rightarrow b \odot T \\
b \odot [S \Leftrightarrow T] &\equiv b \odot S \Leftrightarrow b \odot T
\end{aligned}
$$

When $\{x_1, \ldots, x_n\} = \alpha S \setminus \alpha T$

$$
b \odot [S \restriction T] \equiv \exists b_{\odot} [x_1 : e_1; \cdots; x_n : e_n] \bullet b \odot (S \wedge T)
$$

$$
b \odot S \backslash [x_1, \ldots, x_n] \equiv \exists b_{\odot} [x_1 : e_1; \cdots; x_n : e_n] \bullet b \odot S
$$

When $\alpha S \cap (\phi b \odot T \cup \alpha b) = \varnothing$:

$$
\begin{aligned}
b \odot [\forall S \bullet T] &\equiv \forall b \odot S \bullet b \odot T \\
b \odot [\exists S \bullet T] &\equiv \exists b \odot S \bullet b \odot T \\
b \odot [\exists_1 S \bullet T] &\equiv \exists_1 b \odot S \bullet b \odot T
\end{aligned}
$$

$$
\begin{aligned}
b \odot [S[x_1/y_1, \ldots, x_n/y_n]] &\equiv \\
b \odot [S ; T] &\equiv \\
b \odot [S^*] &\equiv
\end{aligned}
$$

A.2.3 Expressions

$$
\begin{aligned}
b \circledcirc x &\equiv x \quad \text{when } x \notin \alpha b \\
\{ b, x := e \} \circledcirc x &\equiv e \\
\{ b, x := e \} \circledcirc y &\equiv \{ b \} \circledcirc e \\
b \circledcirc x[y] &\equiv \\
b \circledcirc i &\equiv i \\
b \circledcirc z &\equiv z \\
b \circledcirc \{e_1, \ldots, e_n\} &\equiv \{b \circledcirc e_1, \ldots, b \circledcirc e_n\} \\
\text{When } \alpha b \cap \phi e \subseteq \alpha S\text{:} & \\
b \circledcirc \{S \bullet e\} &\equiv \{b \odot S \bullet e\} \\
b \circledcirc (\mu S \bullet e) &\equiv \mu b \circledcirc S \bullet e \\
\text{When } \alpha b \cap \alpha S \cap \phi e = \varnothing \text{ and } \alpha S \cap \phi b = \varnothing\text{:} & \\
b \circledcirc \{S \bullet e\} &\equiv \{b \odot S \bullet b \odot e\} \\
b \circledcirc (\mu S \bullet e) &\equiv \mu b \circledcirc S \bullet b \circledcirc e \\
b \circledcirc \mathbb{P} s &\equiv \mathbb{P} b \circledcirc s \\
b \circledcirc ((e_1, \ldots, e_n)) &\equiv (b \circledcirc e_1, \ldots, b \circledcirc e_n) \\
b \circledcirc (s_1 \times \cdots \times s_n) &\equiv b \circledcirc s_1 \times \cdots \times b \circledcirc s_n \\
b \circledcirc (e.i) &\equiv (b \circledcirc e).i \\
b \circledcirc \{ x_1 := e_1, \ldots, x_n := e_n \} &\equiv \{ x_1 := b \circledcirc e_1, \ldots, x_n := b \circledcirc e_n \} \\
\text{When } \alpha b \cap \alpha S = \varnothing & \\
b \circledcirc \theta S &\equiv \theta b \circledcirc S \\
\text{When } \alpha b = \alpha S & \\
b \circledcirc \theta S &\equiv b \\
b_1 \circledcirc b.x &\equiv (b_1 \circledcirc b).x \\
b \circledcirc (fe) &\equiv (b \circledcirc f)(b \circledcirc e) \\
b \circledcirc (\textbf{if } P \textbf{ then } e_1 \textbf{ else } e_2 \textbf{ fi}) &\equiv \textbf{if } b \odot P \textbf{ then } b \circledcirc e_1 \textbf{ else } b \circledcirc e_2 \\
b_1 \circledcirc (b \odot e) &\equiv \\
\{ x := v \} \odot (\{ x := u \} \odot e) &\equiv \{ x := \{ x := v \} \odot u \} \odot e \\
\{ y := v \} \odot (\{ x := u \} \odot e) &\equiv \{ x := \{ y := v \} \odot u \} \odot \\
& \qquad (\{ y := v \} \odot e) \\
& \quad \text{when } x \notin \phi v.
\end{aligned}
$$

A.2.4 Schema expressions

$$
\begin{aligned}
b \circledcirc [x_1, \ldots x_n : s] &\equiv [x_1, \ldots x_n : b \circledcirc s] \\
\text{When } \alpha b \cap \alpha S = \varnothing & \\
b \circledcirc [S \mid P] &\cong [b \odot S \mid b \odot P] \\
\text{When } \alpha b \cap \Phi P \subseteq \alpha S & \\
b \circledcirc [S \mid P] &\cong [b \odot S \mid P] \\
b \circledcirc [\neg S] &\equiv [\neg b \odot S] \\
b \circledcirc [S \wedge T] &\cong [b \odot S \wedge b \odot T] \\
b \circledcirc [S \vee T] &\cong [b \odot S \vee b \odot T] \\
b \circledcirc [S \Rightarrow T] &\equiv [b \odot S \Rightarrow b \odot T] \\
b \circledcirc [S \Leftrightarrow T] &\equiv [b \odot S \Leftrightarrow b \odot T] \\
b \circledcirc [S \restriction T] &\cong [b \circledcirc S \restriction b \circledcirc T] \\
b \circledcirc S \backslash [x_1, \ldots, x_n] &\equiv (b \circledcirc S) \backslash [x_1, \ldots, x_n] \\
b \circledcirc [\forall S \bullet T] &\equiv [\forall b \odot S \bullet b \odot T] \\
b \circledcirc [\exists S \bullet T] &\equiv [\exists b \odot S \bullet b \odot T] \\
b \circledcirc [\exists_1 S \bullet T] &\cong [\exists_1 b \odot S \bullet b \odot T] \\
b \circledcirc [S[x_1/y_1, \ldots, x_n/y_n]] &\equiv [b \circledcirc S[x_1/y_1, \ldots, x_n/y_n]] \\
b \circledcirc [S \fatsemi T] &\equiv [b \circledcirc S \fatsemi b \circledcirc T] \\
b \circledcirc [S^q] &\equiv [(b \circledcirc S)^q]
\end{aligned}
$$

A.3 Inference rules

A.3.1 Structural rules

Assumption rules

$$\frac{}{\Gamma \ddagger P \vdash P} \quad \textit{AssumPred}$$

$$\frac{}{\Gamma \ddagger x := e \vdash x = e} \quad \textit{AssumDefin}\ (\mathbf{wf}(x := e))$$

$$\frac{}{\Gamma \ddagger x : s \vdash x \in s} \quad \textit{AssumDecl}\ (\mathbf{wf}(x : s))$$

$$\frac{}{\Gamma \ddagger S \vdash S} \quad \textit{SchemaAss}\ (\alpha S \cap \phi S = \varnothing)$$

Paragraph and thinning rules

$$\frac{P \wedge R \vdash Q}{P \ddagger R \vdash Q} \quad \uparrow\downarrow \textit{PredConj}$$

$$\frac{\Gamma \ddagger S \ddagger P \vdash Q}{\Gamma \ddagger S \mid P \vdash Q} \quad \uparrow\downarrow \textit{SchPred}$$

$$\frac{\Gamma \vdash P}{\Pi \ddagger \Gamma \vdash P} \quad \textit{Thinl}$$

$$\frac{\Gamma \vdash P}{\Gamma \ddagger \Pi \vdash P} \quad \textit{Thinr}\ (\alpha \Pi \cap \phi P = \varnothing)$$

$$\frac{\Gamma_1 \ddagger \Gamma_2 \ddagger \Pi \vdash P}{\Gamma_1 \ddagger \Pi \ddagger \Gamma_2 \vdash P} \quad \textit{Shift} \left(\begin{array}{l} \alpha \Gamma_2 \cap \phi \Pi = \varnothing \\ \alpha \Pi \cap \phi \Gamma_2 = \varnothing \end{array} \right)$$

A.3.2 Equality and substitution

$$\frac{}{\Gamma \vdash e = e} \quad \textit{Refl}$$

$$\frac{\Gamma \vdash u = e}{\Gamma \vdash e = u} \quad \textit{Symm}$$

$$\frac{\Gamma \ddagger u = e \vdash v = e}{\Gamma \ddagger u = e \vdash v = u} \quad \textit{Trans}$$

$$\frac{\Gamma \ddagger b \vdash P}{\Gamma \vdash b \odot P} \quad \uparrow\downarrow \textit{UseBind}$$

$$\frac{\Gamma \ddagger b \vdash u = e}{\Gamma \vdash u = b \odot e} \quad \uparrow\downarrow \textit{EquBind}\ (\alpha b \cap \phi u = \varnothing)$$

A.3.3 Propositional calculus

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \; \mathit{AndI}$$

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P} \; \mathit{AndEr}$$

$$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q} \; \mathit{AndEl}$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q} \; \mathit{OrIr}$$

$$\frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q} \; \mathit{OrIl}$$

$$\frac{\Gamma \vdash P \vee Q \qquad \Gamma \ddagger P \vdash R \qquad \Gamma \ddagger Q \vdash R}{\Gamma \vdash R} \; \mathit{OrE}$$

$$\frac{\Gamma \ddagger P \vdash Q}{\Gamma \vdash P \Rightarrow Q} \; \mathit{impI}$$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash P \Rightarrow Q}{\Gamma \vdash Q} \; \mathit{impE}$$

$$\frac{\Gamma \vdash \mathbf{false}}{\Gamma \vdash P} \; \mathit{falseE}$$

$$\frac{\Gamma \ddagger \neg P \vdash \mathbf{false}}{\Gamma \vdash P} \; \mathit{notE}$$

$$\frac{\Gamma \vdash P \Rightarrow \mathbf{false}}{\Gamma \vdash \neg P} \; \uparrow\downarrow \mathit{notDef}$$

$$\frac{}{\Gamma \vdash \mathbf{false} \Rightarrow P} \; \mathit{falseDef}$$

$$\frac{}{\Gamma \vdash P \Rightarrow \mathbf{true}} \; \mathit{trueDef}$$

$$\frac{\Gamma \vdash P \Rightarrow Q \wedge Q \Rightarrow P}{\Gamma \vdash P \Leftrightarrow Q} \; \uparrow\downarrow \mathit{iffDef}$$

A.3.4 Quantifier rules

$$\frac{\Gamma \ddagger S \vdash P}{\Gamma \vdash \forall S \bullet P} \quad \uparrow\downarrow \mathit{AllI}$$

$$\frac{\Gamma \vdash \forall S \bullet P \qquad \Gamma \vdash b \in S}{\Gamma \vdash b \odot P} \quad \mathit{AllE}$$

$$\frac{\Gamma \vdash b \odot P \qquad \Gamma \vdash b \in S}{\Gamma \vdash \exists S \bullet P} \quad \mathit{ExistsI}$$

$$\frac{\Gamma \vdash \exists S \bullet P \qquad \Gamma \ddagger S \ddagger P \vdash Q}{\Gamma \vdash Q} \quad \mathit{ExistsE}$$

$$\frac{\Gamma \vdash \exists S \bullet P \wedge \mathit{SOMETHING}}{\Gamma \vdash \exists_1 S \bullet P} \quad \uparrow\downarrow \mathit{UniqExists}$$

A.3.5 Expression rules

Sets

$$\frac{\Gamma \ddagger [x] T \vdash \{ \{ x := s \}_{\odot} T \bullet y \} \in \mathbb{P} e}{\Gamma \ddagger [x] T \vdash y_{[S]} \in e} \quad \uparrow\downarrow \mathit{GenMem}\ (y \in \alpha T, \mathbf{wf}\ T)$$

$$\frac{\Gamma \vdash (\forall x : s \bullet x \in t) \wedge (\forall x : t \bullet x \in s)}{\Gamma \vdash s = t} \quad \uparrow\downarrow \mathit{Seteq}\ (x \notin \phi s \cup \phi t)$$

$$\frac{\Gamma \vdash v = e_1 \vee \ldots \vee v = e_n}{\Gamma \vdash v \in \{e_1, \ldots, e_n\}} \quad \uparrow\downarrow \mathit{Extmem}$$

$$\frac{\Gamma \vdash \exists S \bullet e = u}{\Gamma \vdash e \in \{S \bullet u\}} \quad \uparrow\downarrow \mathit{Setcomp}\ (\phi e \cap \alpha S = \varnothing)$$

$$\frac{\Gamma \vdash \forall x : t \bullet x \in s}{\Gamma \vdash t \in \mathbb{P} s} \quad \uparrow\downarrow \mathit{Powerset}\ (x \notin \phi s)$$

Cartesian products

$$\frac{\Gamma \vdash v = e_i}{\Gamma \vdash v = (e_1, \ldots, e_n).i} \quad \mathit{Tupleequ}\ (1 \leq i \leq n)$$

$$\frac{\Gamma \vdash u.1 \in s_1 \wedge \ldots \wedge u.n \in s_n}{\Gamma \vdash u \in s_1 \times \cdots \times s_n} \quad \uparrow\downarrow \mathit{Prodmem}$$

$$\frac{\Gamma \vdash u = (e_1, \ldots, e_n)}{\Gamma \vdash u.1 = e_1 \wedge \ldots \wedge u.n = e_n} \quad \uparrow\downarrow \mathit{Tuplesel}$$

Labelled products

$$\frac{}{\Gamma \vdash \langle\!|\, x_1 := e_1, \ldots, x_n := e_n \,|\!\rangle\, x_i = e_i} \quad \mathit{BindEqu}\ (1 \le i \le n)$$

$$\frac{\Gamma \vdash y.x_1 = e_1 \wedge \ldots \wedge y.x_n = e_n}{\Gamma \vdash y = \langle\!|\, x_1 := e_1, \ldots, x_n := e_n \,|\!\rangle} \quad \uparrow\downarrow \mathit{BindSel}$$

$$\frac{}{\Gamma \natural b \vdash x = b.x} \quad \mathit{BindMem}\ (x \in \alpha b, \mathbf{wf}\, b)$$

Schemas

$$\frac{\Gamma \vdash e.x_1 = x_1 \wedge \cdots \wedge e.x_n = x_n \qquad \Gamma \vdash S}{\Gamma \vdash e = \theta S} \quad \mathit{ThetaEqu}$$

$$\frac{\vdash \langle\!|\, x_1 := x_1, \ldots, x_n := x_n \,|\!\rangle \in S}{\vdash S} \quad \uparrow\downarrow \mathit{BindSch}\ (\alpha S = \{x_1, \ldots, x_n\})$$

Description

$$\frac{\Gamma \vdash (e, u) \in f \wedge \forall y : f \bullet (y.1 = e) \Rightarrow (y.2 = ey)}{\Gamma \vdash u = fe} \quad \uparrow\downarrow \mathit{FunctApp}\ (y \notin \phi e \cup \phi u)$$

$$\frac{\begin{array}{c}\Gamma \vdash e \in s \\ \Gamma \vdash \langle\!|\, x := e \,|\!\rangle \odot P \\ \Gamma \natural y : s \vdash \langle\!|\, x := y \,|\!\rangle \odot P \Rightarrow y = e\end{array}}{\Gamma \vdash e = \mu x : s \mid P} \quad \mathit{DefnDescr}$$

$$\frac{\Gamma \vdash P \Rightarrow e_1 = e \wedge \neg P \Rightarrow e_2 = e}{\Gamma \vdash \mathbf{if}\ P\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2\ \mathbf{fi} = e} \quad \uparrow\downarrow \mathit{IfThenElse}$$

$$\frac{\Gamma \vdash v = \langle\!|\, x := u \,|\!\rangle \odot e}{\Gamma \natural x := u \vdash v = e} \quad \uparrow\downarrow \mathit{Usedef}\ (x \notin \phi v)$$

A.3.6 Schema calculus

$$\frac{\Gamma \vdash u \in [x_1 : s_1; \ldots; x_n : s_n]}{\Gamma \vdash u.x_1 \in s_1 \wedge \ldots \wedge u.x_n \in s_n} \quad \uparrow\downarrow \mathit{BindProd}$$

$$\frac{\Gamma \vdash b \in S \wedge b \odot P}{\Gamma \vdash b \in [S \mid P]} \quad \uparrow\downarrow \mathit{SchemaMem}$$

$$\frac{\Gamma \vdash b \odot S}{\Gamma \vdash b \in S} \quad \uparrow\downarrow \mathit{SchBindMem}\ (\mathbf{wf}\, S)$$

$$\frac{\Gamma \vdash b \odot [b \mathrel{\_} S]}{\Gamma \vdash b \in S} \quad \uparrow\downarrow SchBindMem' \ (\mathbf{wf}\, b)$$

$$\frac{\Gamma \vdash \neg\, b \odot S}{\Gamma \vdash b \in \neg\, S} \quad \uparrow\downarrow SNot \ (\mathbf{wf}\, S)$$

$$\frac{\Gamma \vdash b \odot S \wedge b \odot T}{\Gamma \vdash b \in (S \wedge T)} \quad \uparrow\downarrow SAnd \ (\mathbf{wf}\, S \wedge T)$$

$$\frac{\Gamma \vdash b \odot S \vee b \odot T}{\Gamma \vdash b \in (S \vee T)} \quad \uparrow\downarrow Sor \ (\mathbf{wf}\, S \vee T)$$

$$\frac{\Gamma \vdash b \odot S \Rightarrow b \odot T}{\Gamma \vdash b \in (S \Rightarrow T)} \quad \uparrow\downarrow SImp \ (\mathbf{wf}\, S \Rightarrow T)$$

$$\frac{\Gamma \vdash b \odot S \Leftrightarrow b \odot T}{\Gamma \vdash b \in (S \Leftrightarrow T)} \quad \uparrow\downarrow SIff \ (\mathbf{wf}\, S \Leftrightarrow T)$$

$$\frac{\Gamma \vdash \exists S \bullet b \odot T}{\Gamma \vdash b \in \exists S \bullet T} \quad \uparrow\downarrow SExists \ (\phi T \cap (\alpha b \cup \alpha S) = \varnothing)$$

$$\frac{\Gamma \vdash \forall S \bullet b \odot T}{\Gamma \vdash b \in \forall S \bullet T} \quad \uparrow\downarrow SAll \ (\phi T \cap (\alpha b \cup \alpha S) = \varnothing)$$

$$\frac{\Gamma \vdash \exists_1 S \bullet b \odot T}{\Gamma \vdash b \in \exists_1 S \bullet T} \quad \uparrow\downarrow SUniqExists \ (\phi T \cap (\alpha b \cup \alpha S) = \varnothing)$$

$$\frac{\Gamma \vdash b \in \exists x_1 : s_1; \ \cdots; \ x_n : s_n \bullet S}{\Gamma \vdash b \in S \backslash [x_1, \ldots, x_n]} \quad \uparrow\downarrow SHide$$

$$\frac{\Gamma \vdash b \in (S \wedge T) \backslash [x_1, \ldots, x_n]}{\Gamma \vdash b \in S \upharpoonright T} \quad \uparrow\downarrow SProj \ (\alpha S \setminus \alpha T = \{x_1, \ldots, x_n\})$$

$$\frac{SOMETHING}{\Gamma \vdash S \fatsemi T} \quad \uparrow\downarrow SComp$$

$$\frac{SOMETHING}{\Gamma \vdash S^{?}} \quad \uparrow\downarrow SDecor$$