Derivation of Z Refinement Proof Rules

Forwards and backwards rules incorporating input/output refinement

David Cooper, Susan Stepney, and Jim Woodcock

Contents

Chapter 1

Introduction

The traditional set of data refinement rules for Z are stated in [Spivey 1992, section 5.6]. These are sufficient to prove many data refinements that occur in practice, but not all. In the late 1990s we performed the specification and full refinement proof of a large, industrial scale application, that of an Electronic Purse [Stepney *et al.* 2000]. In the course of this work we discovered that the traditional rules were not sufficient to prove our particular refinement. In particular, the traditional obligations assume the use of a 'forward' (or 'downward') simulation, which was inappropriate for our application. We developed a more widely applicable set of Z data refinement proof obligations, for use on our project. These obligations allow both 'forward' and 'backward' simulations [Woodcock & Davies 1996, chapter 16], and also allow non-trivial initialisation, finalisation, and input/output refinement [Stepney *et al.* 1998].

This monograph, originally produced as part of the Electronic Purse development project, provides the full derivation of these rules for refinement in Z. It covers both the traditional forwards and the new backwards refinement proof rules, and the input/output refinement rules.

The purpose of this monograph is threefold:

1. to make explicit the simplifying assumptions that go into deriving these rules
2. to provide enough working that the source of our derivations is clear
3. to provide enough working that other rules can more readily be derived when making different assumptions

This monograph should be read in conjunction with [Woodcock & Davies 1996, chapter 16]; wherever the detail of a derivation or an explanation of a step is given there, it is only referenced here. We present the specific notations used as toolkit definitions in appendix A.

1.1   Structure of the derivations

We define our notion of refinement in terms of programs as sequences of operations, where the operations are defined as relations between states. We then explain in section 2 that this definition reduces to two sets of sufficient (but not necessary) proof rules on individual operations, under the constraint that the operations, initialisations, finalisations and retrieves are total. This is done in the relational calculus.

In section 3 we derive modifications to these proof rules that allow us to relax the constraint on totality of operations (although we still need total initialisation, finalisation and retrieve).

In section 4 we consider special cases of relations where it is possible to identify 'inputs' and 'outputs', and preserve sequences of inputs yet to be consumed and outputs already produced as part of the state. When the retrieves respect this division, we derive further rules that unwind the sequences of inputs and outputs and refer only to single inputs and outputs.

All this work is in the relational calculus. In section 5 we recast the proof rules in the more familiar Z form. This yields the proof rules used in the Electronic Purse case study [Stepney *et al.* 2000].

We have extracted some key lemmas and their proofs in appendix B. (When referenced in the text, these appear in **bold type**.)

Chapter 2

The relational view of refinement

[Woodcock & Davies 1996, chapter 16] go into some detail explaining the origin of refinement and the definition we use. We summarise here.

2.1 What is refinement?

Refinement is concerned with the circumstances under which one data type, $\mathcal{A}$, can usefully be replaced by another more concrete data type, $\mathcal{C}$. The data types may behave differently, but the theory of refinement gives us a tool to identify those aspects of the behaviour to be regarded as important, and allows us to ascertain whether two data types are equivalent *in respect of these aspects*.

The definition we use assumes that we have some 'global' world G from which we can move to either of our data types, then perform manipulations in the data types, and then return to the global world. We compare the results achieved via the two routes. We say that one data type is a refinement of another if the results achievable with it are all allowable results achievable with the other.

We must have the global world and a way to move between it and the data types so we can compare like with like. The map from global world to the data type is called *initialisation*, and from the data type to the global world, *finalisation*.

We frequently have a series of refinements, towards progressively more concrete data types. Near the top of a series of refinements, the global world is usually very similar to the abstract world, and so the initilisation and finalisation operations appear trivial. But as the refinement proceeds in a series of steps, the 'abstract' world of one step is the 'concrete' world of the preceding step, and the corresponding initialisation and finalisation may be non-trivial.

2.2   The definition of relational refinement

The property of interest is the total global to global relation $gg : G \leftrightarrow G$ captured by the data types. The concrete data type will usually capture a smaller relation, as it refines away non-determinism.

The global world is G, the abstract world is A, and the concrete is C. ai and ci are the total abstract and concrete initialisation relations

$$ai : G \leftrightarrow A$$
$$ci : G \leftrightarrow C$$

af and cf are the total abstract and concrete finalisation relations

$$af : A \leftrightarrow G$$
$$cf : C \leftrightarrow G$$

ao and co are the total abstract and concrete operation relations.

$$ao : A \leftrightarrow A$$
$$co : C \leftrightarrow C$$

So the data type $\mathcal{A}$ is the tuple (A, ai, af, ao), and $\mathcal{C}$ is (C, ci, cf, co).

A *program* is a sequence of operations, starting with initialisation, then a finite number of operations, then a finalisation. The relation gg is that defined by all the programs.

The concrete data type refines the abstract data type precisely when the corresponding total global to global relation is a subset:

$$\mathcal{A} \sqsubseteq \mathcal{C} \Leftrightarrow gg_{\mathcal{C}} \subseteq gg_{\mathcal{A}}$$

2.3   The rules of relational refinement

In order to prove refinement without having to reason over the space of all programs, [He Jifeng *et al.* 1986] show that two sets of *unwound* proof rules are sufficient to prove refinement, the so-called forward and backward rules (sometimes called downward and upward rules). They express these rules in the relational calculus, with the constraint that all the relations are total.

The rules from [He Jifeng *et al.* 1986] are as follows.

The forward rules use a retrieve relation r from abstract to concrete; r need not be total. The backward rules use a retrieve relation s from concrete to abstract; s is required to be total by the finalisation proof rule.

$$r : A \leftrightarrow C$$
$$s : C \leftrightarrow A$$

[He Jifeng *et al.* 1986]'s relation refinement proof rules are

Forward

$$ci \subseteq ai \mathbin{\raisebox{0.1ex}{\tiny o}_9} r$$

$$r \mathbin{\raisebox{0.1ex}{\tiny o}_9} cf \subseteq af$$

$$r \mathbin{\raisebox{0.1ex}{\tiny o}_9} co \subseteq ao \mathbin{\raisebox{0.1ex}{\tiny o}_9} r$$

Backward

$$ci \mathbin{\raisebox{0.1ex}{\tiny o}_9} s \subseteq ai$$

$$cf \subseteq s \mathbin{\raisebox{0.1ex}{\tiny o}_9} af$$

$$co \mathbin{\raisebox{0.1ex}{\tiny o}_9} s \subseteq s \mathbin{\raisebox{0.1ex}{\tiny o}_9} ao$$

In the following sections of this monograph we show how these rules can be applied to partial relations, incorporate structure into the state to model inputs and outputs, and recast the rules in a Z state and operations style.

Chapter 3

Untotalising (relaxing)

[He Jifeng *et al.* 1986]'s rules are appropriate for total operations. This section gives equivalent rules when this constraint is relaxed. Initialisation and finalisation are still required to be total. A non-total initialisation can be made total by restricting the global world to the domain of the initialisation. Finalisation needs to be total, because it is always possible to 'pull the plug' on a computation.

The derivation is covered in detail [Woodcock & Davies 1996], so here we just state the results.

3.1   Choice of totalisation

Various sets, X, are augmented with a distinguished element, $\perp$, denoting undefinedness, to give sets $X^\perp$. The non-total operations are *totalised* by defining some appropriate behaviour on the rest of their augmented domain. The retrieve relation is *lifted* to include this distinguished element. The totalised operations and lifted retrieve relations are substituted into the [He Jifeng *et al.* 1986] rules, and, after some algebra, rules appropriate to the non-total operations result.

The totalisation of the operations and lifting of the retrieve relation chosen in [Woodcock & Davies 1996], and used throughout the rest of this monograph, are

$$\overset{\bullet}{\rho} = (X^\perp \times Y^\perp) \oplus \rho$$

$$\overset{\circ}{r} = r \cup (\perp \times Y^\perp)$$

So a partial relation is allowed *any* behaviour outside its domain, and the undefined element retrieves to all elements. This interpretation of partiality leads to the 'widening the precondition', or 'non-blocking', refinement rules.

Although this is the most common interpretation of behaviour outside the precondition, other interpretations are possible. For example, [Bolton 1998] choses

$$\overset{\blacktriangledown}{\rho} = (X^{\perp} \times \{\perp\}) \oplus \rho$$

$$\overset{\triangledown}{r} = r \cup (\perp \times \perp)$$

Here a partial relation is allowed *no* behaviour outside its domain, and the undefined element retrieves to the undefined element only. This interpretation of partiality leads to the different 'firing condition', or 'blocking', refinement rules. We do not consider this form of refinement further here.

3.2 Untotalising forward simulations

From [Woodcock & Davies 1996, Table 16.1], we have the following rules.

Untotalising forward initialisation: when ai and ci are total, we have

$$ci \subseteq ai \mathbin{\,\raise.1ex\hbox{$\scriptstyle\circ$}\hbox{$\scriptstyle 9$}\,} r$$

Untotalising forward finalisation: when af and cf are total, we have

$$r \mathbin{\,\raise.1ex\hbox{$\scriptstyle\circ$}\hbox{$\scriptstyle 9$}\,} cf \subseteq af$$

Untotalising forward correctness: when ao and co are partial, we have

$$(\mathrm{dom}\ ao) \lhd r \mathbin{\,\raise.1ex\hbox{$\scriptstyle\circ$}\hbox{$\scriptstyle 9$}\,} co \subseteq ao \mathbin{\,\raise.1ex\hbox{$\scriptstyle\circ$}\hbox{$\scriptstyle 9$}\,} r$$
$$\wedge\ \mathrm{ran}((\mathrm{dom}\ ao) \lhd r) \subseteq \mathrm{dom}\ co$$

This is broken at the conjunction and treated in future as two rules, the first being the *correctness* condition and the second the *applicability* rule.

3.3 Untotalising backward retrieval

From [Woodcock & Davies 1996, Table 16.1], we have the following rules.

Untotalising backward initialisation: when ai and ci are total, we have

$$ci \mathbin{\,\raise.1ex\hbox{$\scriptstyle\circ$}\hbox{$\scriptstyle 9$}\,} s \subseteq ai$$

Untotalising backward finalisation: when af and cf are total, we have

$$cf \subseteq s \mathbin{\,{}_9^{\,\circ}} af$$

Untotalising backward correctness: when ao and co are partial, we have

$$\mathrm{dom}(s \rhd (\mathrm{dom}\ ao)) \lhd co \mathbin{\,{}_9^{\,\circ}} s \subseteq s \mathbin{\,{}_9^{\,\circ}} ao$$
$$\wedge\ \overline{\mathrm{dom}\ co} \subseteq \mathrm{dom}(s \rhd (\mathrm{dom}\ ao))$$

This is broken at the conjunction and treated in future as two rules, the first being the *correctness* condition and the second the *applicability* rule.

Chapter 4

Incorporating the computational model (unwinding)

The rules expressed so far involve operations that relate some unstructured before state to some unstructured after state. In this section we put structure on the state space in order to model inputs and outputs (following [Woodcock & Davies 1996]). We show how the proof rules can be simplified to refer to the inputs and outputs explicitly.

Z refinement is traditionally viewed as preserving the sequence of inputs and outputs, but not necessarily the individual states passed through. Also, inputs and outputs are not traditionally refined, which leads to the proof rules expressed in [Spivey 1992]. By going back to a more abstract definition we have the freedom to allow more refinements than traditionally, but we are forced to make an explicit choice of what is preserved and what is not.

We choose here to continue with the traditional approach of preserving the full sequence of inputs and outputs. In addition we choose to preserve some properties about the final state. Where we diverge from the treatment in [Woodcock & Davies 1996] is that we also allow inputs and outputs to be refined, which requires us to specify the relation between the abstract and concrete versions.

4.1   State space structure

Our global, abstract and concrete state spaces must therefore be rich enough to hold state, input sequences and output sequences. To avoid confusion over the word 'state', we use *state* to refer to the traditional Z state, and *world* to refer to the state space upon which the relations are defined. So, to preserve inputs and outputs, we must embed the sequence of yet-to-be-consumed inputs and the

already-produced outputs into the worlds, along with the actual Z state. Our global world therefore consists of a global state of type GS, a sequence of global inputs each of type GI, and a sequence of global outputs each of type GO. We represent this as

$$G == GS \times (\text{seq } GI \times \text{seq } GO)$$

The abstract and concrete worlds have a similar structure:

$$A == AS \times (\text{seq } AI \times \text{seq } AO)$$
$$C == CS \times (\text{seq } CI \times \text{seq } CO)$$

We build up the full initialisation, finalisation, operation and retrieve relations from relations defined on the separate parts of the worlds. Starting with the retrieve relations, for the forward rules these are

$$\begin{array}{|l}
\rho : AS \leftrightarrow CS \\
\iota : AI \leftrightarrow CI \\
o : AO \leftrightarrow CO
\end{array}$$

For the backward rules these are (note the redefinition of ι and o to be relations the other way around)

$$\begin{array}{|l}
\sigma : CS \leftrightarrow AS \\
\iota_b : CI \leftrightarrow AI \\
o_b : CO \leftrightarrow AO
\end{array}$$

We also have relations that map between our global and abstract/concrete worlds:

$$\begin{array}{|l}
gcs : GS \leftrightarrow CS \\
gas : GS \leftrightarrow AS \\
gci : GI \leftrightarrow CI \\
gai : GI \leftrightarrow AI \\
gco : GO \leftrightarrow CO \\
gao : GO \leftrightarrow AO
\end{array}$$

The abstract and concrete operations are built up from relations from state-and-input to state-and-output (and hence are more Z-like)

$$\begin{array}{|l}
\alpha : AS \times AI \leftrightarrow AS \times AO \\
\gamma : CS \times CI \leftrightarrow CS \times CO
\end{array}$$

We choose to model state initialisation with a relation that ignores its arguments, thus in effect just defining a set of allowed abstract and concrete initial states.

$$cis == GS \times \{\, cs : CS \mid \text{constraint on } cs \,\}$$
$$ais == GS \times \{\, as : AS \mid \text{constraint on } as \,\}$$

We could allow *cis* and *ais* to pay attention to their arguments, and this is certainly expressible in Z, but there seems little use in normal Z specifications, so we use the above simplification here.

With these pieces, we can now write our definitions of the relations used in the rules to date. The retrievals are different for the forward and backward rules. For the forward rules the retrieval relation is

$$r == \rho \parallel (\widehat{\iota} \parallel \widehat{o})$$

(see appendix A for definitions of these operators) and for the backward rules is

$$s == \sigma \parallel (\widehat{\iota_b} \parallel \widehat{o_b})$$

For both rules the remaining relations are

$$ci == cis \parallel (\widehat{gci} \parallel empty[GO, CO])$$

$$ai == ais \parallel (\widehat{gai} \parallel empty[GO, AO])$$

$$cf == gcs^\sim \parallel (empty[CI, GI] \parallel \widehat{gco^\sim})$$

$$af == gas^\sim \parallel (empty[AI, GI] \parallel \widehat{gao^\sim})$$

$$co == split \,\mathbin{\substack{\circ\\\circ}}\, (\gamma \parallel id) \,\mathbin{\substack{\circ\\\circ}}\, merge$$

$$ao == split \,\mathbin{\substack{\circ\\\circ}}\, (\alpha \parallel id) \,\mathbin{\substack{\circ\\\circ}}\, merge$$

We now incorporate these definitions into the proof rules from chapter 3, and simplify the rules to refer to the sub-pieces only.

4.2   Assumptions

It is worth noting the nature of the assumptions implicit in all these definitions.

1. Retrieval can be split into three *independent* parts: state, input and output. This is why we describe the retrieve relation as the parallel composition of three parts. This means that there must be sufficient information in each part to do its own retrieval — outputs cannot be retrieved differently depending upon state, for example.

2. Retrieval, initialisation and finalisation of inputs and outputs are point-wise on the sequences. That is, despite the presence of a sequence of inputs or outputs, each element is handled separately. This is the meaning of our use of lifted relations.

3. Although future inputs and past outputs exist in the world, the operations are defined only on the current state and current input, and lead to the new state and new output only. This is captured by the use of the *split* and *merge* in the definitions of the operations, and by the fact that the unused parts of the input and output sequences are passed forward with an identity relation.

4. Initialisation ignores its state argument, so all 'programs' are non-deterministically started into one of a set of allowed states. Only the inputs to be consumed are fixed. This is described by the form of *cis* and *ais* and by the choice of *empty* in the definition of *ci* and *ai* to start the output sequences off empty.

5. Finalisation preserves the whole sequence of outputs, but only the last state. This is described by the use of *empty* in the definition of *cf* and *af* to discard the input sequence, but the use of *gcs* and *gas* to preserve the state and $\widehat{gco}$ and $\widehat{gao}$ to preserve the output sequences.

4.3   Computational model in forward simulation

4.3.1   Computational model in forward initialisation

The proof rule after untotalising for initialisation is

$$ci \subseteq ai \,\mathbin{\vphantom{)}^\circ_\circ}\, r$$

We expand the definitions of the relations from section 4.

$$cis \parallel (\widehat{gci} \parallel empty[GO, CO]) \subseteq (ais \parallel (\widehat{gai} \parallel empty[GO, AO])) \,\mathbin{\vphantom{)}^\circ_\circ}\, (\rho \parallel (\widehat{\iota} \parallel \widehat{o}))$$

Parallel and sequential composition *abide* [Woodcock & Davies 1996], so we can reorder

$$cis \parallel (\widehat{gci} \parallel empty[GO, CO]) \subseteq (ais \,\mathring{,}\, \rho) \parallel ((\widehat{gai} \parallel empty[GO, AO]) \,\mathring{,}\, (\widehat{\iota} \parallel \widehat{o}))$$

and again

$$cis \parallel (\widehat{gci} \parallel empty[GO, CO]) \subseteq (ais \,\mathring{,}\, \rho) \parallel ((\widehat{gai} \,\mathring{,}\, \widehat{\iota}) \parallel (empty[GO, AO] \,\mathring{,}\, \widehat{o}))$$

The three parallel strands can be extracted as individual predicates (from the definition of $\parallel$)

$$cis \subseteq ais \,\mathring{,}\, \rho$$
$$\widehat{gci} \subseteq \widehat{gai} \,\mathring{,}\, \widehat{\iota}$$
$$empty[GO, CO] \subseteq empty[GO, AO] \,\mathring{,}\, \widehat{o}$$

The last of these is always true, because $\widehat{o}$ takes the empty sequence to the empty sequence (by the definition of lifting), and the range of *empty* is only the empty sequence (in fact, this shows that equality holds), and so can be discarded. By lemma **lift** we can un-lift the second, yielding two proof rules, one for state and one for input:

$$cis \subseteq ais \,\mathring{,}\, \rho$$
$$gci \subseteq gai \,\mathring{,}\, \iota$$

4.3.2 Computational model in forward finalisation

After untotalising, the finalisation proof rule is

$$r \,\mathring{,}\, cf \subseteq af$$

We expand using the definitions from section 4.1.

$$(\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \,\mathring{,}\, (gcs^{\sim} \parallel (empty[CI, GI] \parallel \widehat{gco^{\sim}}))$$
$$\subseteq gas^{\sim} \parallel (empty[AI, GI] \parallel \widehat{gao^{\sim}})$$

Parallel and sequential composition abide so we can reorder

$$(\rho \,\mathring{,}\, gcs^{\sim}) \parallel ((\widehat{\iota} \parallel \widehat{o}) \,\mathring{,}\, (empty[CI, GI] \parallel \widehat{gco^{\sim}}))$$
$$\subseteq gas^{\sim} \parallel (empty[AI, GI] \parallel \widehat{gao^{\sim}})$$

and again

$$(\rho \fatsemi gcs^\sim) \parallel ((\widehat{\imath} \fatsemi empty[CI, GI]) \parallel (\widehat{o} \fatsemi \widehat{gco^\sim}))$$
$$\subseteq gas^\sim \parallel (empty[AI, GI] \parallel \widehat{gao^\sim})$$

The three parallel strands can be separated as three predicates

$$\rho \fatsemi gcs^\sim \subseteq gas^\sim$$
$$\widehat{\imath} \fatsemi empty[CI, GI] \subseteq empty[AI, GI]$$
$$\widehat{o} \fatsemi \widehat{gco^\sim} \subseteq \widehat{gao^\sim}$$

The second of these is always true, because, as *empty* maps all arguments to the empty sequence, the sequential composition with $\widehat{\imath}$ on the left hand side can only reduce the domain of *empty*. This predicate can therefore be discarded. By lemma **lift** the third can be un-lifted, yielding two proof rules, one for state and one for output.

$$\rho \fatsemi gcs^\sim \subseteq gas^\sim$$
$$o \fatsemi gco^\sim \subseteq gao^\sim$$

4.3.3   Computational model in forward applicability

When untotalising the correctness condition, an additional proof rule is generated, which we call the applicability rule.

$$\mathrm{ran}((\mathrm{dom}\, ao) \lhd r) \subseteq \mathrm{dom}\, co$$

We start by writing these as set comprehensions. The left hand side can be written

$$\mathrm{ran}((\mathrm{dom}\, ao) \lhd r) =$$
$$\mathrm{ran}\{\, sa : AS;\ ia : \mathrm{seq}\, AI;\ oa : \mathrm{seq}\, AO;\ sc : CS;\ ic : \mathrm{seq}\, CI;\ oc : \mathrm{seq}\, CO \mid$$
$$(sa, (ia, oa)) \mapsto (sc, (ic, oc)) \in r$$
$$\wedge\ (sa, (ia, oa)) \in \mathrm{dom}\, ao \bullet$$
$$(sa, (ia, oa)) \mapsto (sc, (ic, oc)) \}$$

We can expand dom *ao* based on the definition of *ao*, making use of lemma **sequential-dom** to push the domain through the sequential composition.

$$\mathrm{dom}\, ao$$
$$= \mathrm{dom}(split \fatsemi (\alpha \parallel id) \fatsemi merge)$$
$$= split^\sim (\!\mid \mathrm{dom}((\alpha \parallel id) \fatsemi merge) \mid\!)$$

merge is total, so it can be dropped

$$= split^\sim (\!| \ \mathrm{dom}(\alpha \parallel id) \ |\!)$$

Expand as a set

$$
\begin{aligned}
= \{ \ &sa : AS; \ ia : \mathrm{seq}\, AI; \ oa : \mathrm{seq}\, AO \mid \\
&\quad (sa, (ia, oa)) \in \mathrm{dom}\, split \\
&\quad \wedge \ split(sa, (ia, oa)) \in \mathrm{dom}(\alpha \parallel id) \bullet \\
&(sa, (ia, oa)) \ \}
\end{aligned}
$$

Expand the definition of *split*

$$
\begin{aligned}
= \{ \ &sa : AS; \ ia : \mathrm{seq}\, AI; \ oa : \mathrm{seq}\, AO \mid \\
&\quad ia \neq \langle \, \rangle \\
&\quad \wedge \ ((sa, head\ ia), (tail\ ia, oa)) \in \mathrm{dom}(\alpha \parallel id) \bullet \\
&(sa, (ia, oa)) \ \}
\end{aligned}
$$

Drop the parallel branch, which adds no constraints

$$
\begin{aligned}
= \{ \ &sa : AS; \ ia : \mathrm{seq}\, AI; \ oa : \mathrm{seq}\, AO \mid ia \neq \langle \, \rangle \wedge (sa, head\ ia) \in \mathrm{dom}\, \alpha \bullet \\
&(sa, (ia, oa)) \ \}
\end{aligned}
$$

Plugging this expression for dom *ao* into the expansion of the left hand side of the original inequality, we get

$$
\begin{aligned}
\mathrm{ran}((\mathrm{dom}\, ao) \lhd r) = \\
\mathrm{ran}\{ \ &sa : AS; \ ia : \mathrm{seq}\, AI; \ oa : \mathrm{seq}\, AO; \ sc : CS; \ ic : \mathrm{seq}\, CI; \ oc : \mathrm{seq}\, CO \mid \\
&\quad (sa, (ia, oa)) \mapsto (sc, (ic, oc)) \in r \\
&\quad \wedge \ ia \neq \langle \, \rangle \wedge (sa, head\ ia) \in \mathrm{dom}\, \alpha \bullet \\
&(sa, (ia, oa)) \mapsto (sc, (ic, oc)) \ \}
\end{aligned}
$$

Taking the range just alters the form of the constructing term

$$
\begin{aligned}
\mathrm{ran}((\mathrm{dom}\, ao) \lhd r) = \\
\{ \ &sa : AS; \ ia : \mathrm{seq}\, AI; \ oa : \mathrm{seq}\, AO; \ sc : CS; \ ic : \mathrm{seq}\, CI; \ oc : \mathrm{seq}\, CO \mid \\
&\quad (sa, (ia, oa)) \mapsto (sc, (ic, oc)) \in r \\
&\wedge \ ia \neq \langle \, \rangle \wedge (sa, head\ ia) \in \mathrm{dom}\, \alpha \bullet \\
&(sc, (ic, oc)) \ \}
\end{aligned}
$$

We do a similar thing with the right hand side, giving us

$$
\begin{aligned}
\mathrm{dom}\, co = \\
\{ \ &sc : CS; \ ic : \mathrm{seq}\, CI; \ oc : \mathrm{seq}\, CO \mid \\
&\quad ic \neq \langle \, \rangle \wedge (sc, head\ ic) \in \mathrm{dom}\, \gamma \bullet \\
&(sc, (ic, oc)) \ \}
\end{aligned}
$$

We now write the fact that the left hand side is a subset of the right hand side by quantifying over the elements of the left hand side, and showing that membership of the left hand side set implies membership of the right hand side set.

$$\forall \, sa : AS; \, ia : \text{seq} \, AI; \, oa : \text{seq} \, AO; \, sc : CS; \, ic : \text{seq} \, CI; \, oc : \text{seq} \, CO \mid$$
$$(sa, (ia, oa)) \mapsto (sc, (ic, oc)) \in r$$
$$\wedge \, ia \neq \langle \, \rangle \wedge (sa, head \, ia) \in \text{dom} \, \alpha \, \bullet$$
$$ic \neq \langle \, \rangle \wedge (sc, head \, ic) \in \text{dom} \, \gamma$$

We expand out the definition of r.

$$\forall \, sa : AS; \, ia : \text{seq} \, AI; \, oa : \text{seq} \, AO; \, sc : CS; \, ic : \text{seq} \, CI; \, oc : \text{seq} \, CO \mid$$
$$sa \mapsto sc \in \rho \wedge ia \mapsto ic \in \widehat{\iota} \wedge oa \mapsto oc \in \widehat{o}$$
$$\wedge \, ia \neq \langle \, \rangle \wedge (sa, head \, ia) \in \text{dom} \, \alpha \, \bullet$$
$$ic \neq \langle \, \rangle \wedge (sc, head \, ic) \in \text{dom} \, \gamma$$

Because lifted relations always map empty sequences to empty sequences, the implication concerning the non-emptiness of the input sequences is discharged automatically.

$$\forall \, sa : AS; \, ia : \text{seq} \, AI; \, oa : \text{seq} \, AO; \, sc : CS; \, ic : \text{seq} \, CI; \, oc : \text{seq} \, CO \mid$$
$$sa \mapsto sc \in \rho \wedge ia \mapsto ic \in \widehat{\iota} \wedge oa \mapsto oc \in \widehat{o}$$
$$\wedge \, (sa, head \, ia) \in \text{dom} \, \alpha \, \bullet$$
$$(sc, head \, ic) \in \text{dom} \, \gamma$$

We discard the elements that are not referred to in the implication (oa and oc) and add single elements of input to draw out the head of the input sequences explicitly

$$\forall \, sa : AS; \, ia : \text{seq} \, AI; \, sc : CS; \, ic : \text{seq} \, CI; \, a? : AI; \, c? : CI \mid$$
$$a? = head \, ia \wedge c? = head \, ic$$
$$\wedge \, sa \mapsto sc \in \rho \wedge ia \mapsto ic \in \widehat{\iota}$$
$$\wedge \, (sa, head \, ia) \in \text{dom} \, \alpha \, \bullet$$
$$(sc, head \, ic) \in \text{dom} \, \gamma$$

We push the property that input sequences are related by $\widehat{\iota}$ down to a property on the individual elements (unlifting $\widehat{\iota}$ in the process), and discard the input sequences themselves.

$$\forall \, sa : AS; \, sc : CS; \, a? : AI; \, c? : CI \mid$$
$$sa \mapsto sc \in \rho \wedge a? \mapsto c? \in \iota$$
$$\wedge \, (sa, a?) \in \text{dom} \, \alpha \, \bullet$$
$$(sc, c?) \in \text{dom} \, \gamma$$

We now write this back in the relational form

$$\text{ran}(\text{dom} \, \alpha \lhd (\rho \parallel \iota)) \subseteq \text{dom} \, \gamma$$

4.3.4   Computational model in forward correctness

Untotalising the correctness proof rule yielded two parts, one of which is handled above as the applicability rule, and the other is handled here as correctness.

$$(\text{dom } ao) \lhd r \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, co \subseteq ao \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, r$$

from section 4.3.3, we write dom ao as a set comprehension

$$\begin{aligned}
\text{dom } ao = \\
\{\, sa : AS;\ ia : \text{seq } AI;\ oa : \text{seq } AO \mid \\
ia \neq \langle\,\rangle \wedge (sa, head\ ia) \in \text{dom } \alpha \bullet \\
(sa, (ia, oa))\,\}
\end{aligned}$$

We expand all the terms using the definitions from section 4.1, and for dom ao above.

$$\begin{aligned}
\{\, sa : AS;\ ia : \text{seq } AI;\ oa : \text{seq } AO \mid ia \neq \langle\,\rangle \wedge (sa, head\ ia) \in \text{dom } \alpha \bullet \\
(sa, (ia, oa))\,\} \\
\lhd (\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, (split \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, (\gamma \parallel id) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, merge) \\
\subseteq \\
(split \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, (\alpha \parallel id) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, merge) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, (\rho \parallel (\widehat{\iota} \parallel \widehat{o}))
\end{aligned}$$

We use the lemmas **split-comm-io** and **merge-comm-io** to shift the *split*s to the left and the *merge*s to the right.

$$\begin{aligned}
\{\, sa : AS;\ ia : \text{seq } AI;\ oa : \text{seq } AO \mid ia \neq \langle\,\rangle \wedge (sa, head\ ia) \in \text{dom } \alpha \bullet \\
(sa, (ia, oa))\,\} \\
\lhd split \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, ((\rho \parallel \iota) \parallel (\widehat{\iota} \parallel \widehat{o})) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, (\gamma \parallel id) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, merge \\
\subseteq \\
split \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, (\alpha \parallel id) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, ((\rho \parallel o) \parallel (\widehat{\iota} \parallel \widehat{o})) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, merge
\end{aligned}$$

The domain restriction can act on the first part only, leaving the *merge* at the ends to be canceled (because *merge* is an injection).

$$\begin{aligned}
\{\, sa : AS;\ ia : \text{seq } AI;\ oa : \text{seq } AO \mid ia \neq \langle\,\rangle \wedge (sa, head\ ia) \in \text{dom } \alpha \bullet \\
(sa, (ia, oa))\,\} \\
\lhd split \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, ((\rho \parallel \iota) \parallel (\widehat{\iota} \parallel \widehat{o})) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, (\gamma \parallel id) \\
\subseteq \\
split \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, (\alpha \parallel id) \,\mathbin{\raise.3ex\hbox{$\mathchar"2039$}}\, ((\rho \parallel o) \parallel (\widehat{\iota} \parallel \widehat{o}))
\end{aligned}$$

We push the domain restriction through the *split* using lemma **restrict-injection**, and the resulting image past the first composition using lemma **sequential-restrict**. We can then cancel the *split*s.

So, the image of the set under *split* is

$$split(\!|\ \{\ sa : AS;\ ia : \operatorname{seq} AI;\ oa : \operatorname{seq} AO \mid ia \neq \langle\rangle \wedge (sa, head\ ia) \in \operatorname{dom}\alpha \bullet$$
$$(sa, (ia, oa))\ \}\ |\!)$$

$$= \{\ sa : AS;\ ia : \operatorname{seq} AI;\ oa : \operatorname{seq} AO \mid ia \neq \langle\rangle \wedge (sa, head\ ia) \in \operatorname{dom}\alpha \bullet$$
$$((sa, head\ ia), (tail\ ia, oa))\ \}$$

$$= \{\ sa : AS;\ ia : \operatorname{seq} AI;\ oa : \operatorname{seq} AO;\ a? : AI \mid (sa, a?) \in \operatorname{dom}\alpha \bullet$$
$$((sa, a?), (ia, oa))\ \}$$

$$= \{\ sa : AS;\ a? : AI \mid (sa, a?) \in \operatorname{dom}\alpha \bullet (sa, a?)\ \} \times (\operatorname{seq} AI \times \operatorname{seq} AO)$$

$$= \operatorname{dom}\alpha \times (\operatorname{seq} AI \times \operatorname{seq} AO)$$

Using this, and canceling *split*, gives us

$$(\operatorname{dom}\alpha \times (\operatorname{seq} AI \times \operatorname{seq} AO)) \lhd ((\rho \parallel \iota) \parallel (\widehat{\iota} \parallel \widehat{o})) \,\fatsemi\, (\gamma \parallel id)$$
$$\subseteq (\alpha \parallel id) \,\fatsemi\, ((\rho \parallel o) \parallel (\widehat{\iota} \parallel \widehat{o}))$$

Using lemma **parallel-restrict**, we can push the restriction inside the first parallel composition.

$$((\operatorname{dom}\alpha \lhd (\rho \parallel \iota)) \parallel ((\operatorname{seq} AI \times \operatorname{seq} AO) \lhd (\widehat{\iota} \parallel \widehat{o}))) \,\fatsemi\, (\gamma \parallel id)$$
$$\subseteq (\alpha \parallel id) \,\fatsemi\, ((\rho \parallel o) \parallel (\widehat{\iota} \parallel \widehat{o}))$$

Parallel and sequential compositions abide, so

$$(\operatorname{dom}\alpha \lhd (\rho \parallel \iota) \,\fatsemi\, \gamma) \parallel ((\operatorname{seq} AI \times \operatorname{seq} AO) \lhd (\widehat{\iota} \parallel \widehat{o}) \,\fatsemi\, id)$$
$$\subseteq (\alpha \,\fatsemi\, (\rho \parallel o)) \parallel (id \,\fatsemi\, (\widehat{\iota} \parallel \widehat{o}))$$

We split the parallel strands into two independent predicates.

$$\operatorname{dom}\alpha \lhd (\rho \parallel \iota) \,\fatsemi\, \gamma \subseteq \alpha \,\fatsemi\, (\rho \parallel o)$$

$$(\operatorname{seq} AI \times \operatorname{seq} AO) \lhd (\widehat{\iota} \parallel \widehat{o}) \,\fatsemi\, id \subseteq id \,\fatsemi\, (\widehat{\iota} \parallel \widehat{o})$$

The second of these is always true, leaving just

$$\operatorname{dom}\alpha \lhd (\rho \parallel \iota) \,\fatsemi\, \gamma \subseteq \alpha \,\fatsemi\, (\rho \parallel o)$$

This is the derived form of the correctness proof rule.

4.3.5   Summary of forward rules after incorporating computational model

Forward initialisation (state) $cis \subseteq ais \mathbin{;} \rho$
Forward initialisation (input) $gci \subseteq gai \mathbin{;} \iota$
Forward finalisation (state) $\rho \mathbin{;} gcs^\sim \subseteq gas^\sim$
Forward finalisation (output) $o \mathbin{;} gco^\sim \subseteq gao^\sim$
Forward applicability $\operatorname{ran}(\operatorname{dom}\alpha \lhd (\rho \parallel \iota)) \subseteq \operatorname{dom}\gamma$
Forward correctness $\operatorname{dom}\alpha \lhd (\rho \parallel \iota) \mathbin{;} \gamma \subseteq \alpha \mathbin{;} (\rho \parallel o)$

4.4   Computational model in backward simulation

4.4.1   Computational model in backward initialisation

The proof rule after untotalising for initialisation is

$$ci \mathbin{;} s \subseteq ai$$

We start by expanding the definitions from section 4.1.

$$(cis \parallel (\widehat{gci} \parallel empty[GO, CO])) \mathbin{;} (\sigma \parallel (\widehat{\iota_b} \parallel \widehat{o_b}))$$
$$\subseteq ais \parallel (\widehat{gai} \parallel empty[GO, AO])$$

Parallel composition and sequential composition abide, so we can reorder this:

$$(cis \mathbin{;} \sigma) \parallel ((\widehat{gci} \parallel empty[GO, CO]) \mathbin{;} (\widehat{\iota_b} \parallel \widehat{o_b}))$$
$$\subseteq ais \parallel (\widehat{gai} \parallel empty[GO, AO])$$

and again

$$(cis \mathbin{;} \sigma) \parallel ((\widehat{gci} \mathbin{;} \widehat{\iota_b}) \parallel (empty[GO, CO] \mathbin{;} \widehat{o_b}))$$
$$\subseteq ais \parallel (\widehat{gai} \parallel empty[GO, AO])$$

These three parallel strands can be separated as three independent predicates

$$cis \mathbin{;} \sigma \subseteq ais$$
$$\widehat{gci} \mathbin{;} \widehat{\iota_b} \subseteq \widehat{gai}$$
$$empty[GO, CO] \mathbin{;} \widehat{o_b} \subseteq empty[GO, AO]$$

The last of these is always true, as all lifted relations map sequences to sequences of the same length, and hence map the empty sequence to the empty sequence. So it can be discarded. The second can be unlifted (see lemma **lift**), yielding two proof rules, one for state and one for input.

$$cis \mathbin{\stackrel{\circ}{\circ}} \sigma \subseteq ais$$

$$gci \mathbin{\stackrel{\circ}{\circ}} \iota_b \subseteq gai$$

4.4.2   Computational model in backward finalisation

The proof rule for finalisation is

$$cf \subseteq s \mathbin{\stackrel{\circ}{\circ}} af$$

Expanding the definitions given in section 4.1 we get

$$gcs^\sim \parallel (empty[CI, GI] \parallel \widehat{gco^\sim})$$
$$\subseteq (\sigma \parallel (\widehat{\iota_b} \parallel \widehat{o_b})) \mathbin{\stackrel{\circ}{\circ}} (gas^\sim \parallel (empty[AI, GI] \parallel \widehat{gao^\sim}))$$

Parallel and sequential composition abide, so we can reorder

$$gcs^\sim \parallel (empty[CI, GI] \parallel \widehat{gco^\sim})$$
$$\subseteq (\sigma \mathbin{\stackrel{\circ}{\circ}} gas^\sim) \parallel ((\widehat{\iota_b} \parallel \widehat{o_b}) \mathbin{\stackrel{\circ}{\circ}} (empty[AI, GI] \parallel \widehat{gao^\sim}))$$

and again

$$gcs^\sim \parallel (empty[CI, GI] \parallel \widehat{gco^\sim})$$
$$\subseteq (\sigma \mathbin{\stackrel{\circ}{\circ}} gas^\sim) \parallel ((\widehat{\iota_b} \mathbin{\stackrel{\circ}{\circ}} empty[AI, GI]) \parallel (\widehat{o_b} \mathbin{\stackrel{\circ}{\circ}} \widehat{gao^\sim}))$$

The three parallel compositions can be separated into three independent predicates

$$gcs^\sim \subseteq \sigma \mathbin{\stackrel{\circ}{\circ}} gas^\sim$$

$$empty[CI, GI] \subseteq \widehat{\iota_b} \mathbin{\stackrel{\circ}{\circ}} empty[AI, GI]$$

$$\widehat{gco^\sim} \subseteq \widehat{o_b} \mathbin{\stackrel{\circ}{\circ}} \widehat{gao^\sim}$$

The second of these is always true as all lifted relations map the empty sequence to the empty sequence. So it can be discarded. The third can be unlifted, yielding

$$gcs^\sim \subseteq \sigma \mathbin{\stackrel{\circ}{\circ}} gas^\sim$$

$$gco^\sim \subseteq o_b \mathbin{\stackrel{\circ}{\circ}} gao^\sim$$

We can split this into two proof rules; one for state and one for outputs.

4.4.3 Computational model in backward applicability

When untotalising the correctness condition, an additional proof rule is generated, which we call the applicability rule.

$$\overline{\operatorname{dom} co} \subseteq \operatorname{dom}(s \rhd (\operatorname{dom} ao))$$

We follow a similar approach to that adopted for the forward rule in section 4.3.3, but omit most of the detail. We can expand out the definition of *co* and *ao* in terms of *split*, *merge*, α and γ. From these we can see that the left hand side can be written

$$\overline{\operatorname{dom} co} = \{\, sc : CS;\ ic : \operatorname{seq} CI;\ oc : \operatorname{seq} CO \mid ic = \langle\,\rangle \vee (sc, head\ ic) \notin \operatorname{dom} \gamma \bullet$$
$$(sc, (ic, oc))\,\}$$

and the right hand side

$$\operatorname{dom}(s \rhd (\operatorname{dom} ao)) =$$
$$\{\, sc : CS;\ ic : \operatorname{seq} CI;\ oc : \operatorname{seq} CO;\ sa : AS;\ ia : \operatorname{seq} AI;\ oa : \operatorname{seq} AO \mid$$
$$(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s$$
$$\wedge\ (ia = \langle\,\rangle \vee (sa, head\ ia) \notin \operatorname{dom} \alpha) \bullet$$
$$(sc, (ic, oc))\,\}$$

Writing the proof rule with quantifiers gives us

$$\forall\, sc : CS;\ ic : \operatorname{seq} CI;\ oc : \operatorname{seq} CO \mid ic = \langle\,\rangle \vee (sc, head\ ic) \notin \operatorname{dom} \gamma \bullet$$
$$\exists\, sa : AS;\ ia : \operatorname{seq} AI;\ oa : \operatorname{seq} AO \bullet$$
$$(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s$$
$$\wedge\ (ia = \langle\,\rangle \vee (sa, head\ ia) \notin \operatorname{dom} \alpha)$$

We can split apart the definition of *s*

$$\forall\, sc : CS;\ ic : \operatorname{seq} CI;\ oc : \operatorname{seq} CO \mid ic = \langle\,\rangle \vee (sc, head\ ic) \notin \operatorname{dom} \gamma \bullet$$
$$\exists\, sa : AS;\ ia : \operatorname{seq} AI;\ oa : \operatorname{seq} AO \bullet$$
$$sc \mapsto sa \in \sigma \wedge ic \mapsto ia \in \widehat{\iota_b} \wedge oc \mapsto oa \in \widehat{o_b}$$
$$\wedge\ (ia = \langle\,\rangle \vee (sa, head\ ia) \notin \operatorname{dom} \alpha)$$

Taking the independent parts of the implication, the quantification over *oc* and *oa* can be discarded because o_b, as part of *s*, is total.

$$\forall\, sc : CS;\ ic : \operatorname{seq} CI \mid ic = \langle\,\rangle \vee (sc, head\ ic) \notin \operatorname{dom} \gamma \bullet$$
$$\exists\, sa : AS;\ ia : \operatorname{seq} AI \bullet$$
$$sc \mapsto sa \in \sigma \wedge ic \mapsto ia \in \widehat{\iota_b}$$
$$\wedge\ (ia = \langle\,\rangle \vee (sa, head\ ia) \notin \operatorname{dom} \alpha)$$

We split the disjunction in the antecedent into two parts

$$(\forall \, sc : CS; \; ic : \text{seq} \, CI \mid ic = \langle \rangle \bullet$$
$$\exists \, sa : AS; \; ia : \text{seq} \, AI \bullet$$
$$sc \mapsto sa \in \sigma \wedge ic \mapsto ia \in \widehat{\iota_b}$$
$$\wedge \, (ia = \langle \rangle \vee (sa, head \, ia) \notin \text{dom} \, \alpha) \,)$$
$$\wedge \, (\forall \, sc : CS; \; ic : \text{seq} \, CI \mid (sc, head \, ic) \notin \text{dom} \, \gamma \bullet$$
$$\exists \, sa : AS; \; ia : \text{seq} \, AI \bullet$$
$$sc \mapsto sa \in \sigma \wedge ic \mapsto ia \in \widehat{\iota_b}$$
$$\wedge \, (ia = \langle \rangle \vee (sa, head \, ia) \notin \text{dom} \, \alpha) \,)$$

The first conjunct is *true*, because σ is total, and so can be discarded. In the second conjunct we introduce new variables into the quantifications as we did at the end of section 4.3.3, unlifting the $\widehat{\iota}$ relation.

$$\forall \, sc : CS; \; c? : CI \mid (sc, c?) \notin \text{dom} \, \gamma \bullet$$
$$\exists \, sa : AS; \; a? : AI \bullet$$
$$sc \mapsto sa \in \sigma \wedge c? \mapsto a? \in \iota_b$$
$$\wedge \, (sa, a?) \notin \text{dom} \, \alpha$$

We can now write this in its relational form

$$\overline{\text{dom} \, \gamma} \subseteq \text{dom}((\sigma \parallel \iota_b) \rhd \text{dom} \, \alpha)$$

4.4.4   Computational model in backward correctness

The proof rule derived for correctness is

$$\text{dom}(s \rhd (\text{dom} \, ao)) \lhd co \,\mathbin{\raise0.2ex\hbox{$\fatsemi$}}\, s \subseteq s \,\mathbin{\raise0.2ex\hbox{$\fatsemi$}}\, ao$$

We expand the definitions of *co*, *ao* and *s*.

$$\text{dom}(s \rhd (\text{dom} \, ao)) \lhd split \,\mathbin{\raise0.2ex\hbox{$\fatsemi$}}\, (\gamma \parallel id) \,\mathbin{\raise0.2ex\hbox{$\fatsemi$}}\, merge \,\mathbin{\raise0.2ex\hbox{$\fatsemi$}}\, (\sigma \parallel (\widehat{\iota_b} \parallel \widehat{o_b}))$$
$$\subseteq (\sigma \parallel (\widehat{\iota_b} \parallel \widehat{o_b})) \,\mathbin{\raise0.2ex\hbox{$\fatsemi$}}\, split \,\mathbin{\raise0.2ex\hbox{$\fatsemi$}}\, (\alpha \parallel id) \,\mathbin{\raise0.2ex\hbox{$\fatsemi$}}\, merge$$

From section 4.4.3 the restricting set can be rewritten

$$\text{dom}(s \rhd (\text{dom} \, ao)) =$$
$$\{ \, sc : CS; \; ic : \text{seq} \, CI; \; oc : \text{seq} \, CO; \; sa : AS; \; ia : \text{seq} \, AI; \; oa : \text{seq} \, AO \mid$$
$$(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s$$
$$\wedge \, (ia = \langle \rangle \vee (sa, head \, ia) \notin \text{dom} \, \alpha) \bullet$$
$$(sc, (ic, oc)) \, \}$$

which we can write as the union of two sets (because of the disjunction):

$$\mathrm{dom}(s \rhd (\mathrm{dom}\ ao)) =$$
$$\{\ sc : CS;\ ic : \mathrm{seq}\ CI;\ oc : \mathrm{seq}\ CO;\ sa : AS;\ ia : \mathrm{seq}\ AI;\ oa : \mathrm{seq}\ AO\ |$$
$$(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s$$
$$\wedge\ ia = \langle\ \rangle\ \bullet$$
$$(sc, (ic, oc))\ \}$$
$$\cup$$
$$\{\ sc : CS;\ ic : \mathrm{seq}\ CI;\ oc : \mathrm{seq}\ CO;\ sa : AS;\ ia : \mathrm{seq}\ AI;\ oa : \mathrm{seq}\ AO\ |$$
$$(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s$$
$$\wedge\ (sa, head\ ia) \notin \mathrm{dom}\ \alpha\ \bullet$$
$$(sc, (ic, oc))\ \}$$

The first of these is not in the domain of *co* (because *co* starts with *split*, which needs an element in the input list to extract as the head) and so can be discarded as it does nothing by being domain subtracted. We now have

$$\{\ sc : CS;\ ic : \mathrm{seq}\ CI;\ oc : \mathrm{seq}\ CO;\ sa : AS;\ ia : \mathrm{seq}\ AI;\ oa : \mathrm{seq}\ AO\ |$$
$$(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s$$
$$\wedge\ (sa, head\ ia) \notin \mathrm{dom}\ \alpha\ \bullet$$
$$(sc, (ic, oc))\ \}$$
$$\lhd split \fatsemi (\gamma \parallel id) \fatsemi merge \fatsemi (\sigma \parallel (\widehat{\iota_b} \parallel \widehat{o_b}))$$
$$\subseteq$$
$$(\sigma \parallel (\widehat{\iota_b} \parallel \widehat{o_b})) \fatsemi split \fatsemi (\alpha \parallel id) \fatsemi merge$$

We use the lemmas **split-comm-io** and **merge-comm-io** to move the *split*s to the left and the *merge*s to the right.

$$\{\ sc : CS;\ ic : \mathrm{seq}\ CI;\ oc : \mathrm{seq}\ CO;\ sa : AS;\ ia : \mathrm{seq}\ AI;\ oa : \mathrm{seq}\ AO\ |$$
$$(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s$$
$$\wedge\ (sa, head\ ia) \notin \mathrm{dom}\ \alpha\ \bullet$$
$$(sc, (ic, oc))\ \}$$
$$\lhd split \fatsemi (\gamma \parallel id) \fatsemi ((\sigma \parallel o_b) \parallel (\widehat{\iota_b} \parallel \widehat{o_b})) \fatsemi merge$$
$$\subseteq split \fatsemi ((\sigma \parallel \iota_b) \parallel (\widehat{\iota_b} \parallel \widehat{o_b})) \fatsemi (\alpha \parallel id) \fatsemi merge$$

We can cancel the *merge* from both sides (as *merge* is an injection).

$$\{\ sc : CS;\ ic : \mathrm{seq}\ CI;\ oc : \mathrm{seq}\ CO;\ sa : AS;\ ia : \mathrm{seq}\ AI;\ oa : \mathrm{seq}\ AO\ |$$
$$(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s$$
$$\wedge\ (sa, head\ ia) \notin \mathrm{dom}\ \alpha\ \bullet$$
$$(sc, (ic, oc))\ \}$$
$$\lhd split \fatsemi (\gamma \parallel id) \fatsemi ((\sigma \parallel o_b) \parallel (\widehat{\iota_b} \parallel \widehat{o_b}))$$
$$\subseteq split \fatsemi ((\sigma \parallel \iota_b) \parallel (\widehat{\iota_b} \parallel \widehat{o_b})) \fatsemi (\alpha \parallel id)$$

We push the domain anti-restriction through the *split* using lemma **restrict-injection**, and the resulting image past the first composition using lemma **sequential-restrict**. We can then cancel the *split*s.

So, the image of the domain restricting set under *split* is

$$
split(\{\, sc : CS;\ ic : \operatorname{seq} CI;\ oc : \operatorname{seq} CO;\ sa : AS;\ ia : \operatorname{seq} AI;\ oa : \operatorname{seq} AO \mid
$$
$$
(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s
$$
$$
\land\ (sa, head\ ia) \notin \operatorname{dom} \alpha \ \bullet
$$
$$
(sc, (ic, oc)) \,\} \,)
$$
$$
= \{\, sc : CS;\ ic : \operatorname{seq} CI;\ oc : \operatorname{seq} CO;\ sa : AS;\ ia : \operatorname{seq} AI;\ oa : \operatorname{seq} AO \mid
$$
$$
(sc, (ic, oc)) \mapsto (sa, (ia, oa)) \in s
$$
$$
\land\ (sa, head\ ia) \notin \operatorname{dom} \alpha \ \bullet
$$
$$
((sc, head\ ic), (tail\ ic, oc)) \,\}
$$
$$
= \{\, sc : CS;\ ic : \operatorname{seq} CI;\ c? : CI;\ oc : \operatorname{seq} CO;
$$
$$
sa : AS;\ ia : \operatorname{seq} AI;\ a? : AI;\ oa : \operatorname{seq} AO \mid
$$
$$
sc \mapsto sa \in \sigma \land ic \mapsto ia \in \widehat{\iota_b} \land c? \mapsto a? \in \iota_b \land oc \mapsto oa \in \widehat{o_b}
$$
$$
\land\ (sa, a?) \notin \operatorname{dom} \alpha \ \bullet
$$
$$
((sc, c?), (ic, oc)) \,\}
$$
$$
= \{\, sc : CS;\ c? : CI;\ sa : AS;\ a? : AI \mid
$$
$$
sc \mapsto sa \in \sigma \land c? \mapsto a? \in \iota_b \land (sa, a?) \notin \operatorname{dom} \alpha \ \bullet
$$
$$
(sc, c?) \,\}
$$
$$
\times (\operatorname{dom} \widehat{\iota_b} \times \operatorname{dom} \widehat{o_b})
$$
$$
= \operatorname{dom}((\sigma \parallel \iota_b) \rhd \operatorname{dom} \alpha) \times (\operatorname{dom} \widehat{\iota_b} \times \operatorname{dom} \widehat{o_b})
$$

So pushing this through and canceling the *split*s gives us

$$
(\operatorname{dom}((\sigma \parallel \iota_b) \rhd \operatorname{dom} \alpha) \times (\operatorname{dom} \widehat{\iota_b} \times \operatorname{dom} \widehat{o_b})) \lhd (\gamma \parallel id) \,\fatsemi\, ((\sigma \parallel o_b) \parallel (\widehat{\iota_b} \parallel \widehat{o_b}))
$$
$$
\subseteq ((\sigma \parallel \iota_b) \parallel (\widehat{\iota_b} \parallel \widehat{o_b})) \,\fatsemi\, (\alpha \parallel id)
$$

Sequential and parallel composition abide, so distribute $\fatsemi$ through $\parallel$ on both sides

$$
(\operatorname{dom}((\sigma \parallel \iota_b) \rhd \operatorname{dom} \alpha) \times (\operatorname{dom} \widehat{\iota_b} \times \operatorname{dom} \widehat{o_b})) \lhd ((\gamma \,\fatsemi\, (\sigma \parallel o_b)) \parallel (id \,\fatsemi\, (\widehat{\iota_b} \parallel \widehat{o_b})))
$$
$$
\subseteq ((\sigma \parallel \iota_b) \,\fatsemi\, \alpha) \parallel ((\widehat{\iota_b} \parallel \widehat{o_b}) \,\fatsemi\, id)
$$

We can split the restriction into two parts (see lemma **parallel-restrict**), and then separate out the parallel compositions

$$
\operatorname{dom}((\sigma \parallel \iota_b) \rhd \operatorname{dom} \alpha) \lhd \gamma \,\fatsemi\, (\sigma \parallel o_b) \subseteq (\sigma \parallel \iota_b) \,\fatsemi\, \alpha
$$
$$
(\operatorname{dom} \widehat{\iota_b} \times \operatorname{dom} \widehat{o_b}) \lhd id \,\fatsemi\, (\widehat{\iota_b} \parallel \widehat{o_b}) \subseteq (\widehat{\iota_b} \parallel \widehat{o_b}) \,\fatsemi\, id
$$

The second part is obviously true (because the sequential compositions with identity can be canceled, and we are left with just a domain restriction of a relation being a subset of the relation itself), leaving just

$$\mathrm{dom}((\sigma \parallel \iota_b) \rhd \mathrm{dom}\,\alpha) \lhd \gamma \,\mathbin{\raise0.3ex\hbox{$\circ$}\mkern-5mu\raise-0.3ex\hbox{$\circ$}}\, (\sigma \parallel o_b) \subseteq (\sigma \parallel \iota_b) \,\mathbin{\raise0.3ex\hbox{$\circ$}\mkern-5mu\raise-0.3ex\hbox{$\circ$}}\, \alpha$$

4.4.5   Summary of backward rules after incorporating computational model

Backward initialisation (state) $cis \,\mathbin{\raise0.3ex\hbox{$\circ$}\mkern-5mu\raise-0.3ex\hbox{$\circ$}}\, \sigma \subseteq ais$

Backward initialisation (input) $gci \,\mathbin{\raise0.3ex\hbox{$\circ$}\mkern-5mu\raise-0.3ex\hbox{$\circ$}}\, \iota_b \subseteq gai$

Backward finalisation (state) $gcs^{\sim} \subseteq \sigma \,\mathbin{\raise0.3ex\hbox{$\circ$}\mkern-5mu\raise-0.3ex\hbox{$\circ$}}\, gas^{\sim}$

Backward finalisation (output) $gco^{\sim} \subseteq o_b \,\mathbin{\raise0.3ex\hbox{$\circ$}\mkern-5mu\raise-0.3ex\hbox{$\circ$}}\, gao^{\sim}$

Backward applicability $\overline{\mathrm{dom}\,\gamma} \subseteq \mathrm{dom}((\sigma \parallel \iota_b) \rhd \mathrm{dom}\,\alpha)$

Backward correctness $\mathrm{dom}((\sigma \parallel \iota_b) \rhd \mathrm{dom}\,\alpha) \lhd \gamma \,\mathbin{\raise0.3ex\hbox{$\circ$}\mkern-5mu\raise-0.3ex\hbox{$\circ$}}\, (\sigma \parallel o_b) \subseteq (\sigma \parallel \iota_b) \,\mathbin{\raise0.3ex\hbox{$\circ$}\mkern-5mu\raise-0.3ex\hbox{$\circ$}}\, \alpha$

Chapter 5

Recasting the rules in Z

In this chapter we derive the familiar Z formulation of the proof rules.

5.1 Recasting Z schemas as relations

To do this, we first explain what a Z schema is in term of the relational calculus. This is normally easy, as a schema that relates two sets of variables can be written as a relation by using set comprehension. For example, a Z operation Op with no inputs or outputs, which explains how before and after states are related (S and S'), can be written as a relation between states thus

$$\{\, Op \bullet \theta S \mapsto \theta S' \,\}$$

We now give Z forms of the relations used so far, using common Z schema names for the state retrieve, initialisation, etc.

$G, g? : GI, g! : GO$	[global state, input and output]
$C, c? : CI, c! : CO$	[concrete state, input and output]
$A, a? : AI, a! : AO$	[abstract state, input and output]

The state, input, and output retrieves:

$R \cong [\, A;\ C \mid \text{constraint}\,]$

$RIn \cong [\, a? : AI;\ c? : CI \mid \text{constraint}\,]$

$ROut \cong [\, a! : AO;\ c! : CO \mid \text{constraint}\,]$

The concrete and abstract operations:

$COp \cong [\, C;\ C';\ c? : CI;\ c! : CO \mid \text{constraint}\,]$

$$AOp \mathrel{\widehat{=}} [\, A;\, A';\, a? : AI;\, a! : AO \mid \text{constraint}\,]$$

The concrete and abstract state and input initialisations:

$$CInitState \mathrel{\widehat{=}} [\, C' \mid \text{constraint}\,]$$
$$AInitState \mathrel{\widehat{=}} [\, A' \mid \text{constraint}\,]$$
$$CInitIn \mathrel{\widehat{=}} [\, c? : CI;\, g? : GI \mid \text{constraint}\,]$$
$$AInitIn \mathrel{\widehat{=}} [\, a? : AI;\, g? : GI \mid \text{constraint}\,]$$

The concrete and abstract state and output finalisations:

$$CFinState \mathrel{\widehat{=}} [\, C;\, G \mid \text{constraint}\,]$$
$$AFinState \mathrel{\widehat{=}} [\, A;\, G \mid \text{constraint}\,]$$
$$CFinOut \mathrel{\widehat{=}} [\, c! : CO;\, g! : GO \mid \text{constraint}\,]$$
$$AFinOut \mathrel{\widehat{=}} [\, a! : AO;\, g! : GO \mid \text{constraint}\,]$$

For forward retrieval we have

$$\rho == \{\, R \bullet \theta A \mapsto \theta C \,\}$$
$$\iota == \{\, RIn \bullet a? \mapsto c? \,\}$$
$$o == \{\, ROut \bullet a! \mapsto c! \,\}$$

and for backward retrieval

$$\sigma == \{\, R \bullet \theta C \mapsto \theta A \,\}$$
$$\iota_b == \{\, RIn \bullet c? \mapsto a? \,\}$$
$$o_b == \{\, ROut \bullet c! \mapsto a! \,\}$$

For both retrievals, we have

$$gcs == \{\, CFinState \bullet \theta G \mapsto \theta C \,\}$$
$$gas == \{\, AFinState \bullet \theta G \mapsto \theta A \,\}$$
$$gci == \{\, CInitIn \bullet g? \mapsto c? \,\}$$
$$gai == \{\, AInitIn \bullet g? \mapsto a? \,\}$$
$$gco == \{\, CFinOut \bullet g! \mapsto c! \,\}$$
$$gao == \{\, AFinOut \bullet g! \mapsto a! \,\}$$

$$\gamma == \{\, COp \bullet (\theta C, c?) \mapsto (\theta C', c!) \,\}$$
$$\alpha == \{\, AOp \bullet (\theta A, a?) \mapsto (\theta A', a!) \,\}$$

In section 4.1 we discussed how initialisation is typically expressed in Z. Note that here we define the set comprehension with the dashed variables, because

traditionally the Z initialisation is regarded as an 'operation' that delivers the initial state from nothing, and hence is a predicate on the after-state.

$$cis == G \times \{\, CInitState \bullet \theta C' \,\}$$
$$ais == G \times \{\, AInitState \bullet \theta A' \,\}$$

5.2   Recasting forward simulations

5.2.1   Recasting forward initialisation

From section 4.3.1 we have the following two proof rules

$$cis \subseteq ais \,\mathring{\,}_{9}\, \rho$$

$$gci \subseteq gai \,\mathring{\,}_{9}\, \iota$$

The first rule is the state initialisation, and the second is the input initialisation.

5.2.1.1   State initialisation

$$cis \subseteq ais \,\mathring{\,}_{9}\, \rho$$

We substitute in the definitions in Z of these relations and obtain the following. (We have chosen to use dashed variables in the set comprehension for R to make the equations simpler later. The choice of names does not change the meaning.)

$$G \times \{\, CInitState \bullet \theta C' \,\}$$
$$\subseteq (G \times \{\, AInitState \bullet \theta A' \,\}) \,\mathring{\,}_{9}\, \{\, R' \bullet \theta A' \mapsto \theta C' \,\}$$

As explained in the appendix in B.9 we can write this using quantifiers:

$$\forall\, G;\ C' \mid CInitState \bullet$$
$$\qquad \exists\, A' \bullet (\theta G, \theta A') \in (G \times \{\, AInitState \bullet \theta A' \,\}) \wedge R'$$

$$\Leftrightarrow$$

$$\forall\, G;\ C' \mid CInitState \bullet \exists\, A' \bullet AInitState \wedge R'$$

$$\Leftrightarrow$$

$$\forall\, C' \mid CInitState \bullet \exists\, A' \bullet AInitState \wedge R'$$

This can be written as a theorem

$$CInitState \vdash \exists\, A' \bullet AInitState \wedge R'$$

5.2.1.2   Input initialisation

$$gci \subseteq gai \,\S\, \iota$$

We substitute in the definitions of these relations in Z.

$$\{\, CInitIn \bullet g? \mapsto c? \,\} \subseteq \{\, AInitIn \bullet g? \mapsto a? \,\} \,\S\, \{\, RIn \bullet a? \mapsto c? \,\}$$

Write as quantifiers

$$\forall\, g? : GI;\, c? : CI \mid CInitIn \bullet \exists\, a? : AI \bullet AInitIn \land RIn$$

Write this as a theorem

$$CInitIn \vdash \exists\, a? : AI \bullet AInitIn \land RIn$$

5.2.2   Recasting forward finalisation

From section 4.3.2 we have two parts of the proof rule

$$\rho \,\S\, gcs^{\sim} \subseteq gas^{\sim}$$
$$o \,\S\, gco^{\sim} \subseteq gao^{\sim}$$

The first rule is the state finalisation, and the second is the output finalisation.

5.2.2.1   State finalisation

$$\rho \,\S\, gcs^{\sim} \subseteq gas^{\sim}$$

Expand the definitions of these relations in Z

$$\{\, R \bullet \theta A \mapsto \theta C \,\} \,\S\, \{\, CFinState \bullet \theta G \mapsto \theta C \,\}^{\sim}$$
$$\subseteq \{\, AFinState \bullet \theta G \mapsto \theta A \,\}^{\sim}$$

As described in the appendix in B.9, we can write these as quantifiers. The fact that the relations are written as inverses does not affect the translation.

$$\forall\, A;\, C;\, G \mid R \land CFinState \bullet AFinState$$

We can then write this as a theorem

$$R;\, CFinState \vdash AFinState$$

5.2.2.2   Ouput finalisation

$$o \,\mathring{\,}\, gco^\sim \subseteq gao^\sim$$

Expand these relations using their definitions in Z

$$\{\, ROut \bullet a! \mapsto c! \,\} \,\mathring{\,}\, \{\, CFinOut \bullet g! \mapsto c! \,\}^\sim$$
$$\subseteq \{\, AFinOut \bullet g! \mapsto a! \,\}^\sim$$

As explained in the appendix in B.9 we can write this in quantifiers

$$\forall\, a! : AO;\; c! : CO;\; g! : GO \mid ROut \wedge CFinOut \bullet AFinOut$$

As a theorem this is

$$ROut;\; CFinOut \vdash AFinOut$$

5.2.3   Recasting forward applicability

From section 4.3.3 we have the proof rule

$$\mathrm{ran}(\mathrm{dom}\,\alpha \lhd (\rho \parallel \iota)) \subseteq \mathrm{dom}\,\gamma$$

The parallel composition of the relations ρ and ι can be represented as a set comprehension in which the two part are combined in a tuple.

$$\mathrm{ran}(\mathrm{dom}\,\alpha \lhd \{\, R;\; RIn \bullet (\theta A, a?) \mapsto (\theta C, c?) \,\}) \subseteq \mathrm{dom}\,\gamma$$

Domain restricting this is equivalent to adding a constraint into the set comprehension. The constraint is that the tuple is in the domain of the abstract operation, which in Z is represented by pre *AOp*.

$$\mathrm{ran}\{\, R;\; RIn \mid \mathrm{pre}\ AOp \bullet (\theta A, a?) \mapsto (\theta C, c?) \,\} \subseteq \mathrm{dom}\,\gamma$$

Taking the range of this set only alters the form of the constructing term.

$$\{\, R;\; RIn \mid \mathrm{pre}\ AOp \bullet (\theta C, c?) \,\} \subseteq \mathrm{dom}\,\gamma$$

We now convert to quantifiers as described in B.9. The domain of γ is exactly the set of all those pairs $(\theta C, c?)$ that allow the concrete operation to proceed. This is captured in the Z predicate pre *COp*.

$$\forall\, A;\; C;\; a? : AI;\; c? : CI \mid R \wedge RIn \wedge \mathrm{pre}\ AOp$$
$$\bullet (\theta C, c?) \in \{\, C;\; c? : CI \mid \mathrm{pre}\ COp \bullet (\theta C, c?) \,\}$$

$$\Leftrightarrow$$

$$\forall\, A;\; C;\; a? : AI;\; c? : CI \mid R \wedge RIn \wedge \mathrm{pre}\ AOp \bullet \mathrm{pre}\ COp$$

We write this as a theorem

$$R;\ RIn\ |\ \text{pre}\ AOp \vdash \text{pre}\ COp$$

5.2.4   Recasting forward correctness

From section 4.3.4 we have the proof rule

$$\text{dom}\ \alpha \lhd (\rho \parallel \iota)\ \text{\small\begingroup\rm \char"9F\endgroup}\ \gamma \subseteq \alpha\ \text{\small\char"9F}\ (\rho \parallel o)$$

We write this as set comprehensions, making use of the ideas used in the derivation above, 5.2.3.

$$\text{dom}\ \alpha \lhd \{\ R;\ RIn \bullet (\theta A, a?) \mapsto (\theta C, c?)\ \}\ \text{\small\char"9F}\ \{\ COp \bullet (\theta C, c?) \mapsto (\theta C', c!)\ \}$$
$$\subseteq \{\ AOp \bullet (\theta A, a?) \mapsto (\theta A', a!)\ \}\ \text{\small\char"9F}\ \{\ R';\ ROut \bullet (\theta A', a!) \mapsto (\theta C', c!)\ \}$$

We can replace the domain expression with a set comprehension, too

$$\{\ AOp\ |\ \text{pre}\ AOp \bullet (\theta A, a?)\ \}$$
$$\lhd \{\ R;\ RIn \bullet (\theta A, a?) \mapsto (\theta C, c?)\ \}\ \text{\small\char"9F}\ \{\ COp \bullet (\theta C, c?) \mapsto (\theta C', c!)\ \}$$
$$\subseteq \{\ AOp \bullet (\theta A, a?) \mapsto (\theta A', a!)\ \}\ \text{\small\char"9F}\ \{\ R';\ ROut \bullet (\theta A', a!) \mapsto (\theta C', c!)\ \}$$

We can push the domain restriction into the definition of the first relation in the composition as a predicate

$$\{\ R;\ RIn\ |\ \text{pre}\ AOp \bullet (\theta A, a?) \mapsto (\theta C, c?)\ \}\ \text{\small\char"9F}\ \{\ COp \bullet (\theta C, c?) \mapsto (\theta C', c!)\ \}$$
$$\subseteq \{\ AOp \bullet (\theta A, a?) \mapsto (\theta A', a!)\ \}\ \text{\small\char"9F}\ \{\ R';\ ROut \bullet (\theta A', a!) \mapsto (\theta C', c!)\ \}$$

Write this using quantifiers

$$\forall A;\ a? : AI;\ C;\ c? : CI;\ C';\ c! : CO\ |\ R \wedge RIn \wedge \text{pre}\ AOp \wedge COp \bullet$$
$$\exists A';\ a! : AO \bullet AOp \wedge R' \wedge ROut$$

which can in turn be written as a theorem

$$R;\ RIn;\ COp\ |\ \text{pre}\ AOp \vdash \exists A';\ a! : AO \bullet AOp \wedge R' \wedge ROut$$

5.3   Recasting backward retrievals

5.3.1   Recasting backward initialisation

From section 4.4.1 we have two parts of the proof rule

$$cis \mathbin{{}^\circ_9} \sigma \subseteq ais$$

$$gci \mathbin{{}^\circ_9} \iota_b \subseteq gai$$

The first rule is the state initialisation, and the second is the input initialisation.

5.3.1.1   State initialisation

$$cis \mathbin{{}^\circ_9} \sigma \subseteq ais$$

We substitute in the definitions in Z of these relations and obtain the following. (We have chosen to use dashed variables in the set comprehension for R, to make the equations simpler later. The choice of names does not change the meaning.)

$$(G \times \{\, CInitState \bullet \theta C' \,\}) \mathbin{{}^\circ_9} \{\, R' \bullet \theta C' \mapsto \theta A' \,\} \subseteq$$
$$(G \times \{\, AInitState \bullet \theta A' \,\})$$

As explained in the appendix in B.9 we can write this using quantifiers.

$$\forall G;\ C';\ A' \mid CInitState \wedge R' \bullet$$
$$(\theta G, \theta A') \in (G \times \{\, AInitState \bullet \theta A' \,\})$$

$$\Leftrightarrow$$

$$\forall G;\ C';\ A' \mid CInitState \wedge R' \bullet AInitState$$

$$\Leftrightarrow$$

$$\forall C';\ A' \mid CInitState \wedge R' \bullet AInitState$$

This can be written as a theorem

$$CInitState;\ R' \vdash AInitState$$

5.3.1.2 Input initialisation

$$gci \mathbin{\raise0.5ex\hbox{$\scriptstyle\circ$}\mkern-5mu\raise-0.5ex\hbox{$\scriptstyle\circ$}} \iota_b \subseteq gai$$

Substitute in the definitions for these relations.

$$\{\, CInitIn \bullet g? \mapsto c? \,\} \mathbin{\raise0.5ex\hbox{$\scriptstyle\circ$}\mkern-5mu\raise-0.5ex\hbox{$\scriptstyle\circ$}} \{\, RIn \bullet c? \mapsto a? \,\} \subseteq \{\, AInitIn \bullet g? \mapsto a? \,\}$$

With quantifiers this is

$$\forall\, g? : GI;\, c? : CI;\, a? : AI \mid CInitIn \wedge RIn \bullet AInitIn$$

which can be written as a theorem as

$$CInitIn;\, RIn \vdash AInitIn$$

5.3.2 Recasting backward finalisation

From section 4.4.2 we have two parts of the proof rule

$$gcs^{\sim} \subseteq \sigma \mathbin{\raise0.5ex\hbox{$\scriptstyle\circ$}\mkern-5mu\raise-0.5ex\hbox{$\scriptstyle\circ$}} gas^{\sim}$$
$$gco^{\sim} \subseteq o_b \mathbin{\raise0.5ex\hbox{$\scriptstyle\circ$}\mkern-5mu\raise-0.5ex\hbox{$\scriptstyle\circ$}} gao^{\sim}$$

The first rule is the state finalisation, and the second is the output finalisation.

5.3.2.1 State finalisation

$$gcs^{\sim} \subseteq \sigma \mathbin{\raise0.5ex\hbox{$\scriptstyle\circ$}\mkern-5mu\raise-0.5ex\hbox{$\scriptstyle\circ$}} gas^{\sim}$$

Expanding with the definitions of these relations

$$\{\, CFinState \bullet \theta G \mapsto \theta C \,\}^{\sim}$$
$$\subseteq \{\, R \bullet \theta C \mapsto \theta A \,\} \mathbin{\raise0.5ex\hbox{$\scriptstyle\circ$}\mkern-5mu\raise-0.5ex\hbox{$\scriptstyle\circ$}} \{\, AFinState \bullet \theta G \mapsto \theta A \,\}^{\sim}$$

As explained in the appendix in B.9, as the sequential composition is on the right hand side, it doesn't go away, and instead becomes an existential quantification. The fact that these are the inverses of relations gets lost when they are converted to quantifiers —it is only important in deciding which variable is quantified over on the right hand side.

$$\forall\, G;\, C \mid CFinState \bullet \exists\, A \bullet R \wedge AFinState$$

which is written

$$CFinState \vdash \exists\, A \bullet R \wedge AFinState$$

5.3.2.2 Output finalisation

$$gco^{\sim} \subseteq o_b \mathbin{\substack{\circ\\\circ}} gao^{\sim}$$

Expanding the definitions of the these relations

$$\{\,CFinOut \bullet g! \mapsto c!\,\}^{\sim} \subseteq \{\,ROut \bullet c! \mapsto a!\,\} \mathbin{\substack{\circ\\\circ}} \{\,AFinOut \bullet g! \mapsto a!\,\}^{\sim}$$

Writing as quantifiers

$$\forall\, g! : GO;\ c! : CO \mid CFinOut \bullet \exists\, a! : AO \bullet ROut \wedge AFinOut$$

and then as a theorem

$$CFinOut \vdash \exists\, a! : AO \bullet ROut \wedge AFinOut$$

5.3.3 Recasting backward applicability

From section 4.4.3 we have the proof rule

$$\overline{\operatorname{dom}\gamma} \subseteq \operatorname{dom}((\sigma \parallel \iota_b) \mathbin{\rhd} \operatorname{dom}\alpha)$$

The parallel composition of the relations σ and ι_b can be represented as a set comprehension in which the two parts are combined in a tuple

$$\overline{\operatorname{dom}\gamma} \subseteq \operatorname{dom}(\{\,R;\ RIn \bullet (\theta C, c?) \mapsto (\theta A, a?)\,\} \mathbin{\rhd} \operatorname{dom}\alpha)$$

Range restricting this is equivalent to adding a constraining predicate to the set comprehension. In this case, the constraint is that the elements are not in the precondition of the abstract operation, which is captured in the predicate pre AOp.

$$\overline{\operatorname{dom}\gamma} \subseteq \operatorname{dom}(\{\,R;\ RIn \mid \neg\ \text{pre } AOp \bullet (\theta C, c?) \mapsto (\theta A, a?)\,\})$$

Taking the domain alters only the form of the constructing term in the set comprehension.

$$\overline{\operatorname{dom}\gamma} \subseteq \{\,R;\ RIn \mid \neg\ \text{pre } AOp \bullet (\theta C, c?)\,\}$$

We now convert the relational inclusion to a universal quantification (as described in B.9), and the complement of the domain of the γ relation to a constraint on the universal

$$\forall\, C;\ c? : CI \mid \neg\ \text{pre } COp \bullet \exists\, A;\ a? : AI \bullet R \wedge RIn \wedge (\neg\ \text{pre } AOp)$$

We now shuffle these quantifiers around to get it into the traditional form

$\forall\, C;\ c?:CI \bullet \neg\ \mathrm{pre}\ COp \Rightarrow (\exists\, A;\ a?:AI \bullet R \wedge RIn \wedge (\neg\ \mathrm{pre}\ AOp)\,)$

$\Leftrightarrow$

$\forall\, C;\ c?:CI \bullet \neg\ (\exists\, A;\ a?:AI \bullet R \wedge RIn \wedge (\neg\ \mathrm{pre}\ AOp)\,) \Rightarrow \mathrm{pre}\ COp$

$\Leftrightarrow$

$\forall\, C;\ c?:CI \bullet (\forall\, A;\ a?:AI \bullet \neg\ (R \wedge RIn) \vee \mathrm{pre}\ AOp\,) \Rightarrow \mathrm{pre}\ COp$

$\Leftrightarrow$

$\forall\, C;\ c?:CI \bullet (\forall\, A;\ a?:AI \bullet (R \wedge RIn) \Rightarrow \mathrm{pre}\ AOp\,) \Rightarrow \mathrm{pre}\ COp$

$\Leftrightarrow$

$\forall\, C;\ c?:CI \bullet (\forall\, A;\ a?:AI \mid (R \wedge RIn) \bullet \mathrm{pre}\ AOp\,) \Rightarrow \mathrm{pre}\ COp$

$\Leftrightarrow$

$\forall\, C;\ c?:CI \mid (\forall\, A;\ a?:AI \mid (R \wedge RIn) \bullet \mathrm{pre}\ AOp\,) \bullet \mathrm{pre}\ COp$

We write this as a theorem

$C;\ c?:CI \mid (\forall\, A;\ a?:AI \mid (R \wedge RIn) \bullet \mathrm{pre}\ AOp\,) \vdash \mathrm{pre}\ COp$

5.3.4   Recasting backward correctness

From section 4.4.4 we have the proof rule

$\mathrm{dom}((\sigma \parallel \iota_b) \rhd \mathrm{dom}\,\alpha) \lhd (\gamma \,\mathbin{\raise0.3ex\hbox{$\scriptstyle\circ$}\kern-0.1em\lower0.3ex\hbox{$\scriptstyle\circ$}}\, (\sigma \parallel o_b)) \subseteq (\sigma \parallel \iota_b) \,\mathbin{\raise0.3ex\hbox{$\scriptstyle\circ$}\kern-0.1em\lower0.3ex\hbox{$\scriptstyle\circ$}}\, \alpha$

From the derivation above (section 5.3.3) we can replace the set used for domain subtraction to yield

$\{\, R;\ RIn \mid \neg\ \mathrm{pre}\ AOp \bullet (\theta C, c?) \,\} \lhd (\gamma \,\mathbin{\raise0.3ex\hbox{$\scriptstyle\circ$}\kern-0.1em\lower0.3ex\hbox{$\scriptstyle\circ$}}\, (\sigma \parallel o_b)) \subseteq (\sigma \parallel \iota_b) \,\mathbin{\raise0.3ex\hbox{$\scriptstyle\circ$}\kern-0.1em\lower0.3ex\hbox{$\scriptstyle\circ$}}\, \alpha$

As we did in section 5.3.3, we construct set comprehensions from the parallel composition of relations, and expand the definitions of all the relations. We choose to use R' for the second relation on the left hand side, as this makes the equations easier.

$\{\, R;\ RIn \mid \neg\ \mathrm{pre}\ AOp \bullet (\theta C, c?) \,\}$
$\qquad \lhd (\{\, COp \bullet (\theta C, c?) \mapsto (\theta C', c!) \,\} \,\mathbin{\raise0.3ex\hbox{$\scriptstyle\circ$}\kern-0.1em\lower0.3ex\hbox{$\scriptstyle\circ$}}\, \{\, R';\ ROut \bullet (\theta C', c!) \mapsto (\theta A', a!) \,\})$
$\subseteq \{\, R;\ RIn \bullet (\theta C, c?) \mapsto (\theta A, a?) \,\} \,\mathbin{\raise0.3ex\hbox{$\scriptstyle\circ$}\kern-0.1em\lower0.3ex\hbox{$\scriptstyle\circ$}}\, \{\, AOp \bullet (\theta A, a?) \mapsto (\theta A', a!) \,\}$

We can slide the domain subtraction inside the definition of the first relation in
the composition, expressing the subtraction as a negative predicate

$$\{\, COp \mid \neg\,(\exists\, A;\ a?:AI \bullet R \wedge RIn \wedge \neg\ \mathrm{pre}\ AOp) \bullet (\theta C, c?) \mapsto (\theta C', c!)\,\}$$
$$\qquad {}_{\,9}^{\,9}\,\{\, R';\ ROut \bullet (\theta C', c!) \mapsto (\theta A', a!)\,\}$$
$$\subseteq \{\, R;\ RIn \bullet (\theta C, c?) \mapsto (\theta A, a?)\,\}\, {}_{\,9}^{\,9}\,\{\, AOp \bullet (\theta A, a?) \mapsto (\theta A', a!)\,\}$$

We now expand this out using quantifiers as described in section B.9

$$\forall\, C;\ C';\ c?:CI;\ c!:CO;\ A';\ a!:AO \mid$$
$$\qquad COp \wedge \neg\,(\exists\, A;\ a?:AI \bullet R \wedge RIn \wedge \neg\ \mathrm{pre}\ AOp\,) \wedge R' \wedge ROut \bullet$$
$$\exists\, A;\ a?:AI \bullet R \wedge RIn \wedge AOp$$

We take the negation inside the existential quantifier, converting this to a univer-
sal. We then convert the predicates that are disjoined within the universal to an
implication, and hence to a constrained universal. This yields

$$\forall\, C;\ C';\ c?:CI;\ c!:CO;\ A';\ a!:AO \mid$$
$$\qquad COp \wedge (\forall\, A;\ a?:AI \mid R \wedge RIn \bullet \mathrm{pre}\ AOp\,) \wedge R' \wedge ROut \bullet$$
$$\exists\, A;\ a?:AI \bullet R \wedge RIn \wedge AOp$$

We can now shuffle this to the traditional form by putting this constraining uni-
versal as early as possible

$$\forall\, C;\ c?:CI \mid (\forall\, A;\ a?:AI \mid R \wedge RIn \bullet \mathrm{pre}\ AOp\,) \bullet$$
$$\qquad \forall\, C';\ c!:CO;\ A';\ a!:AO \mid COp \wedge R' \wedge ROut \bullet$$
$$\qquad\qquad \exists\, A;\ a?:AI \bullet R \wedge RIn \wedge AOp$$

This cannot be simplified by being written as a theorem, so the theorem form just
has the turnstile ($\vdash$) before it.

Appendix A

Toolkit

A.1 Lifting to sequences

We 'lift' relations on sets to relations on sequences of sets as follows

$$
\begin{array}{l}
\rule{6cm}{0.4pt}\,[X, Y]\,\rule{4cm}{0.4pt} \\
\widehat{}: (X \leftrightarrow Y) \rightarrow (\mathrm{seq}\,X \leftrightarrow \mathrm{seq}\,Y) \\
\rule{5cm}{0.4pt} \\
\forall\, r : X \leftrightarrow Y \bullet \\
\quad \widehat{r} = \{\, s : \mathrm{seq}(X \times Y) \mid \mathrm{ran}\,s \subseteq r \bullet s \,\fatsemi\, \mathit{first} \mapsto s \,\fatsemi\, \mathit{second} \,\}
\end{array}
$$

The result of lifting r is a relation between pairs of sequences of equal length. For every pair of sequences in $\widehat{r}$, every pair of corresponding elements belong to r.

$$
\begin{array}{l}
[X, Y]\, r : X \leftrightarrow Y;\ s : \mathrm{seq}\,X;\ t : \mathrm{seq}\,Y \mid s \mapsto t \in \widehat{r} \\
\vdash \\
\# s = \# t \\
\wedge\,(\,\forall\, n : \mathrm{dom}\,s \bullet s\,n \mapsto t\,n \in r\,)
\end{array}
$$

Consider $\widehat{r}$ restricted to singleton sequences. It is clear that

$$
[X, Y]\, r : X \leftrightarrow Y \vdash \{\, x : X;\ y : Y \mid x \mapsto y \in r \bullet \langle x \rangle \mapsto \langle y \rangle \,\} \subseteq \widehat{r}
$$

A.2 Split and Merge

We adapt the definitions of *split* and *merge* from [Woodcock & Davies 1996, section 16.5], which manipulate the worlds to extract individual inputs and outputs.

split extracts the first input from the sequence of inputs and the state, packaging them up for delivery to the Z-like operations α and γ introduced in section 4.1.

$$
\begin{array}{|l}
\hline
[S, I, O] \\\\
\hline
split : S \times (\mathrm{seq}_1\, I \times \mathrm{seq}\, O) \rightarrowtail (S \times I) \times (\mathrm{seq}\, I \times \mathrm{seq}\, O) \\\\
\hline
\forall\, s : S;\ i : \mathrm{seq}_1\, I;\ o : \mathrm{seq}\, O \bullet \\\\
\quad split(s, (i, o)) = ((s, head\ i), (tail\ i, o)) \\\\
\hline
\end{array}
$$

merge does the opposite, taking an output and concatenating it to the end of the sequence of outputs.

$$
\begin{array}{|l}
\hline
[S, I, O] \\\\
\hline
merge : (S \times O) \times (\mathrm{seq}\, I \times \mathrm{seq}\, O) \rightarrowtail S \times (\mathrm{seq}\, I \times \mathrm{seq}\, O) \\\\
\hline
\forall\, s : S;\ o : O;\ i : \mathrm{seq}\, I;\ os : \mathrm{seq}\, O \bullet \\\\
\quad merge((s, o), (i, os)) = (s, (i, os \frown \langle o \rangle)) \\\\
\hline
\end{array}
$$

A.3   Parallel composition and Copy

We also use the definition of 'parallel composition' and 'copy' from [Woodcock & Davies 1996, section 16.5].

Parallel composition $\|$ converts a pair of relations into a relation between pairs.

$$
\begin{array}{|l}
\hline
[W, X, Y, Z] \\\\
\hline
\_ \parallel \_ : (W \leftrightarrow Y) \times (X \leftrightarrow Z) \rightarrow W \times X \leftrightarrow Y \times Z \\\\
\hline
\forall\, r : W \leftrightarrow Y;\ s : X \leftrightarrow Z;\ w : W;\ x : X;\ y : Y;\ z : Z \bullet \\\\
\quad (w, x) \mapsto (y, z) \in r \parallel s \Leftrightarrow w \mapsto y \in r \wedge x \mapsto z \in s \\\\
\hline
\end{array}
$$

cp copies its argument

$$
\begin{array}{|l}
\hline
[X] \\\\
\hline
cp : X \rightarrowtail X \times X \\\\
\hline
\forall\, x : X \bullet cp\ x = (x, x) \\\\
\hline
\end{array}
$$

A.4   Empty sequencies and identity realtion

We have use for an implict generic set that comprises a pair of empty sequences:

$$empty[X, Y] == \text{seq} \varnothing[X] \times \text{seq} \varnothing[Y]$$

We have use for an implict generic identity relation:

$$id[X] == \{\, x : X \bullet x \mapsto x \,\}$$

Where the type can be deduced from context it is permissible to omit the generic argument. For clarity's sake, however, we sometimes choose to include the argument explicitly in these cases.

Appendix B

Lemmas and their proofs

B.1 Commuting *split* around sequential composition

Lemma 1 (split-comm-io) *Even in the presence of input/output refinement, split can 'commute' around sequential composition.*

$$\vdash (\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} split = split \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} ((\rho \parallel \iota) \parallel (\widehat{\iota} \parallel \widehat{o}))$$

Proof: This proof follows the one given in [Woodcock & Davies 1996], which used a simpler form in which some of the relations were the identity. We make use of the definition of *split* given in [Woodcock & Davies 1996] using *cp*.

$$(\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} split$$

$$= (\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \left(cp \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \begin{pmatrix} id \parallel (\mathit{first} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \mathit{head}) \\ \parallel \\ \mathit{second} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (\mathit{tail} \parallel id) \end{pmatrix} \right) \qquad \text{[def]}$$

$$= cp \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \begin{pmatrix} \rho \parallel (\widehat{\iota} \parallel \widehat{o}) \\ \parallel \\ \rho \parallel (\widehat{\iota} \parallel \widehat{o}) \end{pmatrix} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \begin{pmatrix} id \parallel (\mathit{first} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \mathit{head}) \\ \parallel \\ \mathit{second} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (\mathit{tail} \parallel id) \end{pmatrix} \qquad \text{[prop of } cp]$$

$$= cp \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \begin{pmatrix} (\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (id \parallel (\mathit{first} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \mathit{head})) \\ \parallel \\ (\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (\mathit{second} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (\mathit{tail} \parallel id)) \end{pmatrix} \qquad \text{[abide]}$$

$$= cp \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \begin{pmatrix} (\rho \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} id) \parallel ((\widehat{\iota} \parallel \widehat{o}) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \mathit{first} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \mathit{head}) \\ \parallel \\ (\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (\mathit{second} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (\mathit{tail} \parallel id)) \end{pmatrix} \qquad \text{[abide]}$$

$$= cp \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \begin{pmatrix} (id \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \rho) \parallel ((\widehat{\iota} \parallel \widehat{o}) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \mathit{first} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} \mathit{head}) \\ \parallel \\ (\rho \parallel (\widehat{\iota} \parallel \widehat{o})) \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (\mathit{second} \mathbin{\raise.09ex\hbox{\scriptsize$\,;\,$}} (\mathit{tail} \parallel id)) \end{pmatrix} \qquad \text{[id commutes]}$$

$$= cp \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \left(\begin{array}{l} (id \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \rho) \parallel ((\widehat{\iota} \parallel \widehat{o}) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{first} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{head}) \\ \parallel \\ \mathit{second} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\widehat{\iota} \parallel \widehat{o}) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\mathit{tail} \parallel id) \end{array} \right) \qquad [(R \parallel S) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{second} = \mathit{second} \parallel S]$$

$$= cp \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \left(\begin{array}{l} (id \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \rho) \parallel \mathit{first} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \widehat{\iota} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{head}) \\ \parallel \\ \mathit{second} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\widehat{\iota} \parallel \widehat{o}) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\mathit{tail} \parallel id) \end{array} \right) \qquad [(R \parallel S) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{first} = \mathit{first} \parallel R]$$

$$= cp \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \left(\begin{array}{l} (id \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \rho) \parallel \mathit{first} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{head} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \iota) \\ \parallel \\ \mathit{second} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\widehat{\iota} \parallel \widehat{o}) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\mathit{tail} \parallel id) \end{array} \right) \qquad [\text{unlift to commute with head}]$$

$$= cp \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \left(\begin{array}{l} (id \parallel (\mathit{first} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{head})) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\rho \parallel \iota) \\ \parallel \\ \mathit{second} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\widehat{\iota} \parallel \widehat{o}) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\mathit{tail} \parallel id) \end{array} \right) \qquad [\text{abide}]$$

$$= cp \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \left(\begin{array}{l} (id \parallel (\mathit{first} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{head})) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\rho \parallel \iota) \\ \parallel \\ \mathit{second} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\mathit{tail} \parallel id) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\widehat{\iota} \parallel \widehat{o}) \end{array} \right) \qquad [\mathit{tail} \parallel id \text{ commutes}]$$

$$= cp \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \left(\begin{array}{l} (id \parallel (\mathit{first} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{head})) \\ \parallel \\ \mathit{second} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\mathit{tail} \parallel id) \end{array} \right) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \left(\begin{array}{l} \rho \parallel \iota \\ \parallel \\ \widehat{\iota} \parallel \widehat{o} \end{array} \right) \qquad [\text{abide}]$$

$$= \mathit{split} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} ((\rho \parallel \iota) \parallel (\widehat{\iota} \parallel \widehat{o})) \qquad [\text{def}]$$

∎ B.1

B.2   Commuting *merge* around sequential composition

Lemma 2 (merge-comm-io) *Even in the presence of input/output refinement, merge can 'commute' around sequential composition.*

$$\vdash \mathit{merge} \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} (\rho \parallel (\widehat{\iota} \parallel \widehat{o})) = ((\rho \parallel o) \parallel (\widehat{\iota} \parallel \widehat{o})) \mathbin{\raise0.5pt\hbox{$\scriptscriptstyle9$}} \mathit{merge}$$

Proof: Very similar to the proof of **split-comm-io**.

B.3   ∃ to ∀ conversion

Lemma 3 (∃-∀-convert) *An existential quantifier as part of the declaration in a universal quantifier can be brought out into the universal.*

$$\forall\, x : X \mid (\,\exists\, y : Y \bullet Q(x, y)\,) \bullet P(x) \vdash \forall\, x : X;\, y : Y \mid Q(x, y) \bullet P(x)$$

Proof:

$\forall\, x : X \mid (\,\exists\, y : Y \bullet Q(x, y)\,) \bullet P(x)$	[hyp]
$\forall\, x : X \bullet (\,\exists\, y : Y \bullet Q(x, y)\,) \Rightarrow P(x)$	[decl = ⇒]
$\forall\, x : X \bullet \neg\, (\,\exists\, y : Y \bullet Q(x, y)\,) \vee P(x)$	[def of ⇒]
$\forall\, x : X \bullet (\,\forall\, y : Y \bullet \neg\, Q(x, y)\,) \vee P(x)$	[def of ¬ ∃]
$\forall\, x : X \bullet \forall\, y : Y \bullet \neg\, Q(x, y) \vee P(x)$	[move brackets]
$\forall\, x : X;\, y : Y \bullet \neg\, Q(x, y) \vee P(x)$	[combine ∀]
$\forall\, x : X;\, y : Y \bullet Q(x, y) \Rightarrow P(x)$	[def of ⇒]
$\forall\, x : X;\, y : Y \mid Q(x, y) \bullet P(x)$	[decl = ⇒]

■ B.3

B.4   Lifting whole expressions

Lemma 4 (lift) *Sequential composition and set inclusion can be lifted.*

$$\vdash \widehat{\rho \,\mathring{\,}_{9}\, \sigma} = \widehat{\rho} \,\mathring{\,}_{9}\, \widehat{\sigma}$$

$$\vdash \widehat{\rho} \subseteq \widehat{\sigma} \Leftrightarrow \rho \subseteq \sigma$$

Proof: Directly from the definition of lifting.

B.5   Pushing restriction into parallel composition

Lemma 5 (parallel-restrict) *Domain restriction or subtraction with a cross product of a parallel composition can be broken into its parts.*

$$[W, X, Y, Z]$$
$$a : W;\ b : X;\ r : W \leftrightarrow Y;\ s : X \leftrightarrow Z$$
$$\vdash$$
$$(a \times b) \lhd (r \parallel s) = (a \lhd r) \parallel (b \lhd s)$$
$$\wedge\ (a \times b) \ntriangleleft (r \parallel s) = (a \ntriangleleft r) \parallel (b \ntriangleleft s)$$

Proof:

$$(a \times b) \lhd (r \parallel s)$$

$$= (a \times b) \lhd \{\, w : W;\ x : X;\ y : Y;\ z : Z \mid w \mapsto y \in r \wedge x \mapsto z \in s \bullet$$
$$(w, x) \mapsto (y, z) \,\} \qquad\qquad [\text{def of } \parallel]$$

$$= \{\, w : W;\ x : X;\ y : Y;\ z : Z \mid w \mapsto y \in r \wedge x \mapsto z \in s \wedge (w, x) \in a \times b \bullet$$
$$(w, x) \mapsto (y, z) \,\} \qquad\qquad [\text{def of } \lhd]$$

$$= \{\, w : W;\ x : X;\ y : Y;\ z : Z \mid w \mapsto y \in r \wedge x \mapsto z \in s \wedge w \in a \wedge x \in b \bullet$$
$$(w, x) \mapsto (y, z) \,\} \qquad\qquad [\text{def of } \times]$$

$$= (a \lhd r) \parallel (b \lhd s) \qquad\qquad [\text{def of } \parallel \text{ and } \lhd]$$

We argue similarly for $\ntriangleleft$.

■ B.5

B.6   Pushing restriction through an injection

Lemma 6 (restrict-injection) *Domain restriction of an injection is the same as range restriction by the relational image.*

$$[X, Y]$$
$$r : X \rightarrowtail Y;\ a : X$$
$$\vdash$$
$$a \lhd r = r \rhd r(\!| \ a \ |\!)$$
$$\wedge\ a \ntriangleleft r = r \ntriangleright r(\!| \ a \ |\!)$$

Proof:

$r \rhd r(\!| \, a \, |\!)$

$\qquad = \{\, x : X;\, y : Y \mid x \mapsto y \in r \wedge y \in r(\!| \, a \, |\!) \,\}$ [defn $\rhd$]

$\qquad = \{\, x : X;\, y : Y \mid x \mapsto y \in r$

$\qquad\qquad\qquad \wedge\, y \in \{\, x_1 : X;\, y_1 : Y \mid x_1 \mapsto y_1 \in r \wedge x_1 \in a \bullet y_1 \,\} \,\}$ [defn $(\!|\,|\!)$]

$\qquad = \{\, x, x_1 : X;\, y, y_1 : Y \mid x \mapsto y \in r \wedge x_1 \mapsto y_1 \in r \wedge x_1 \in a \wedge y = y_1$

$\qquad\quad\, \bullet\, x \mapsto y \,\}$ [simplify]

$\qquad = \{\, x : X;\, y : Y \mid x \mapsto y \in r \wedge x \in a \,\}$ [hyp $r \in X \rightarrowtail Y$]

$\qquad = a \lhd r$ [defn $\lhd$]

We argue similarly for $\lhd$.

■ B.6

B.7   Pushing restriction through sequential composition

Lemma 7 (sequential-restrict) *Restriction (pseudo-)distributes through sequential composition.*

$\quad [X, Y, Z]$
$\quad r : X \leftrightarrow Y;\, s : Y \leftrightarrow Z;\, a : Y$
$\quad \vdash$
$\quad r \rhd a \,\fatsemi\, s = r \,\fatsemi\, a \lhd s$
$\quad \wedge\, r \rhd a \,\fatsemi\, s = r \,\fatsemi\, a \lhd s$

Proof:

$r \rhd a \,\fatsemi\, s$

$\qquad = (r \,\fatsemi\, \mathrm{id}\, a) \,\fatsemi\, s$ [[Spivey 1992, p98]]

$\qquad = r \,\fatsemi\, (\mathrm{id}\, a \,\fatsemi\, s)$ [[Spivey 1992, p97]]

$\qquad = r \,\fatsemi\, a \lhd s$ [[Spivey 1992, p98]]

We argue similarly for $\lhd$.

■ B.7

B.8   Pushing domain through sequential composition

Lemma 8 (sequential-dom) *Domain of a sequential composition can be pushed past the first relation in the composition using inverse relational image.*

$[X, Y, Z]$
$r : X \leftrightarrow Y; s : Y \leftrightarrow Z$
$\vdash$
$\mathrm{dom}(r \,\mathring{\,}_\mathring{9}\, s) = r^{\sim}(\!|\, \mathrm{dom}\, s\, |\!)$

Proof: This is a law stated in [Spivey 1992, page 101], but we prove it here for completeness.

$\mathrm{dom}(r \,\mathring{\,}_\mathring{9}\, s)$

$\quad = \mathrm{dom}\{\, x : X; y : Y; z : Z \mid x \mapsto y \in r \wedge y \mapsto z \in s \bullet x \mapsto z \,\}$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ [def of $\mathring{\,}_\mathring{9}$]

$\quad = \{\, x : X; y : Y; z : Z \mid x \mapsto y \in r \wedge y \mapsto z \in s \bullet x \,\}$ $\qquad$ [def of dom]

$\quad = \{\, x : X; y : Y \mid x \mapsto y \in r \wedge y \in \mathrm{dom}\, s \bullet x \,\}$ $\qquad$ [def of dom]

$\quad = r^{\sim}(\!|\, \mathrm{dom}\, s\, |\!)$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ [def of image]

■ B.8

B.9   Converting relational inclusion to quantification

While recasting proof rules written in the relational calculus into the equivalent form in Z, we frequently encounter sequential composition and relational inclusion. In this section we show how to recast these in general.

Consider four relations on three sets, defined using some general schema predicates.

$r == \{\, x : X; y : Y \mid R \,\}$
$s == \{\, y : Y; z : Z \mid S \,\}$
$t == \{\, x : X; y : Y \mid T \,\}$
$u == \{\, y : Y; z : Z \mid U \,\}$

B.9.1　Converting a composition

Consider the case where the following general combination holds

$$r \mathbin{\substack{\circ\\\circ}} s \subseteq t \mathbin{\substack{\circ\\\circ}} u$$

We recast this in quantified form.

First, the definition of subset is that all elements on the left hand side are also elements on the right hand side. The elements are the result of the sequential composition of relations, so are pairs drawn from the set $(X \times Z)$.

$$r \mathbin{\substack{\circ\\\circ}} s \subseteq t \mathbin{\substack{\circ\\\circ}} u$$
$$\Leftrightarrow$$
$$\forall x : X; z : Z \mid x \mapsto z \in r \mathbin{\substack{\circ\\\circ}} s \bullet x \mapsto z \in t \mathbin{\substack{\circ\\\circ}} u$$

The sequential composition in the bar part is, by definition, the existence of an intermediate value in Y that acts as a stepping stone for the two relations.

$$\Leftrightarrow$$
$$\forall x : X; z : Z \mid (\exists y : Y \bullet x \mapsto y \in r \land y \mapsto z \in s) \bullet$$
$$x \mapsto z \in t \mathbin{\substack{\circ\\\circ}} u$$

As we have schema predicates that capture the property of a pair being in a relation, we can replace the set membership with these predicates.

$$\Leftrightarrow$$
$$\forall x : X; z : Z \mid (\exists y : Y \bullet R \land S) \bullet x \mapsto z \in t \mathbin{\substack{\circ\\\circ}} u$$

We do an analogous thing with the composition in the dot part.

$$\Leftrightarrow$$
$$\forall x : X; z : Z \mid (\exists y : Y \bullet R \land S) \bullet (\exists y : Y \bullet T \land U)$$

Using lemma ∃-∀-**convert** this becomes

$$\Leftrightarrow$$
$$\forall x : X; y : Y; z : Z \mid R \land S \bullet (\exists y : Y \bullet T \land U)$$

Note that the y in the existential (and referred to in T and U) is different from the y in the universal.

B.9.2   Handling inverses

If any of the predicates are inverses, these disappear when converted to set comprehensions or quantifiers. For example, consider

$$s^\sim \mathbin{\mathchoice{}{}{}{}_9^\circ} r^\sim \subseteq u^\sim \mathbin{\mathchoice{}{}{}{}_9^\circ} t^\sim$$

written using quantifiers. We still use the predicates S, R, U and T, which define the forward relations. The first step looks the same, but with the universal over pairs the other way around.

$$s^\sim \mathbin{\mathchoice{}{}{}{}_9^\circ} r^\sim \subseteq u^\sim \mathbin{\mathchoice{}{}{}{}_9^\circ} t^\sim$$
$$\Leftrightarrow$$
$$\forall\, x : X; \, z : Z \mid z \mapsto x \in (s^\sim \mathbin{\mathchoice{}{}{}{}_9^\circ} r^\sim) \bullet z \mapsto x \in (u^\sim \mathbin{\mathchoice{}{}{}{}_9^\circ} t^\sim)$$

We continue as before

$$\Leftrightarrow$$
$$\forall\, x : X; \, z : Z \mid (\exists\, y : Y \bullet z \mapsto y \in s^\sim \wedge y \mapsto x \in r^\sim) \bullet$$
$$z \mapsto x \in u^\sim \mathbin{\mathchoice{}{}{}{}_9^\circ} t^\sim$$

Given the definition of the inverse of a relation, we can swap the variables in the pairs

$$\Leftrightarrow$$
$$\forall\, x : X; \, z : Z \mid (\exists\, y : Y \bullet y \mapsto z \in s \wedge x \mapsto y \in r) \bullet$$
$$z \mapsto x \in u^\sim \mathbin{\mathchoice{}{}{}{}_9^\circ} t^\sim$$

and continue as before, giving

$$\Leftrightarrow$$
$$\forall\, x : X; \, z : Z \mid (\exists\, y : Y \bullet S \wedge R) \bullet (\exists\, y : Y \bullet U \wedge T)$$

and hence

$$\Leftrightarrow$$
$$\forall\, x : X; \, y : Y; \, z : Z \mid R \wedge S \bullet (\exists\, y : Y \bullet T \wedge U)$$

exactly as before.

Appendix C

Summary of derived proof rules

The Z proof rules derived in this monograph are as follows.

C.1   Forward

Forward initialisation (state)

$CInitState \vdash \exists A' \bullet AInitState \wedge R'$

Forward initialisation (input)

$CInitIn \vdash \exists a? : AI \bullet AInitIn \wedge RIn$

Forward finalisation (state)

$R; CFinState \vdash AFinState$

Forward finalisation (output)

$ROut; CFinOut \vdash AFinOut$

Forward applicability

$R; RIn \mid \text{pre } AOp \vdash \text{pre } COp$

Forward correctness

$R; RIn; COp \mid \text{pre } AOp \vdash \exists A'; a! : AO \bullet AOp \wedge R' \wedge ROut$

C.2 Backward

Backward initialisation (state)

$$CInitState; R' \vdash AInitState$$

Backward initialisation (input)

$$CInitIn; RIn \vdash AInitIn$$

Backward finalisation (state)

$$CFinState \vdash \exists A \bullet R \wedge AFinState$$

Backward finalisation (output)

$$CFinOut \vdash \exists a! : AO \bullet ROut \wedge AFinOut$$

Backward applicability

$$C; c? : CI \mid (\forall A; a? : AI \mid (R \wedge RIn) \bullet \text{pre } AOp) \vdash \text{pre } COp$$

Backward correctness

$$\vdash \forall C; c? : CI \mid (\forall A; a? : AI \mid R \wedge RIn \bullet \text{pre } AOp) \bullet$$
$$\forall C'; c! : CO; A'; a! : AO \mid COp \wedge R' \wedge ROut \bullet$$
$$\exists A; a? : AI \bullet R \wedge RIn \wedge AOp$$

Appendix D

Bibliography

[Bolton 1998]
     Christie Bolton. Objects, Processes and States. MSc thesis, Programming Research Group, Oxford University Computing Laboratory, 1998.

[He Jifeng *et al.* 1986]
     He Jifeng, C. A. R. Hoare, and Jeff W. Sanders. Data refinement refined (resumé). In *ESOP'86*, number 213 in Lecture Notes in Computer Science, pages 187–196. Springer Verlag, 1986.

[Spivey 1992]
     J. Michael Spivey. *The Z Notation: a Reference Manual*. Prentice Hall, 2nd edition, 1992.

[Stepney *et al.* 1998]
     Susan Stepney, David Cooper, and Jim Woodcock. More Powerful Z Data Refinement: pushing the state of the art in industrial refinement. In Jonathan P. Bowen, Andreas Fett, and Michael G. Hinchey, editors, *ZUM '98: The Z Formal Specification Notation*, 11th International Conference of Z Users, Berlin, September 1998, volume 1493 of Lecture Notes in Computer Science, pages 284–307. Springer-Verlag, 1998.

[Stepney *et al.* 2000]
     Susan Stepney, David Cooper, and Jim Woodcock. An Electronic Purse: specification, refinement and proof. Technical Monograph PRG-126, Programming Research Group, Oxford University Computing Laboratory, 2000.

[Woodcock & Davies 1996]
     Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.