Calculating upward and downward simulations of state-based specifications

John Derrick and Eerke Boiten
Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Phone: + 44 1227 764000, Email: J.Derrick@ukc.ac.uk.)

Abstract

This paper concerns calculational methods of refinement in state-based specification languages. Data refinement is a well established technique for transforming specifications of abstract data types into ones which are closer to an eventual implementation. The conditions under which a transformation is a correct refinement are encapsulated into two simulation rules: downward and upward simulations.

One approach to refining an abstract system is to specify the concrete data type, and then attempt to verify that it is a valid refinement of the abstract type. An alternative approach is to *calculate* the concrete specification based upon the abstract specification and a retrieve relation which links the abstract and concrete states. In this paper we generalise existing calculational methods for downward simulations and derive similar results for upward simulations; we also document their use and application in a particular specification language, namely Z.

Keywords: Refinement; State-based systems; Z; Calculating refinements.

1 Introduction

This paper concerns methods by which we can calculate refinements of systems specified in state-based specification languages such as Z [8], B [1] and VDM [6]. These state-based languages have gained a certain amount of acceptance in the software community as an industrial strength formal method. As a canonical example, we will concentrate on Z in this paper, although the remarks we make apply equally to similar languages. Z is a state-based language whose specifications are written using set theory and first order logic. Abstract data types are specified in Z using the so called "state plus operations" style, where a collection of operations describe changes to the state space. The state space, initialisation and operations are described as *schemas*, and the schema calculus has proved to be an enduring structuring mechanism for specifying complex systems. These schemas, and the operations that they represent, can be understood as (total or partial) relations on the underlying state space.

In addition to specifying a system, we might also wish to develop, or *refine*, it further. This idea of data refinement is a well established technique for transforming specifications of abstract data types into ones which are closer to an eventual implementation. The conditions under which a development is a correct refinement are encapsulated into two refinement (or simulation) rules: downward and upward simulations [9]. These refinement rules are known to be sound and jointly complete, that is any upward or downward simulation is a valid refinement, and any refinement can be proved correct by application of appropriate upward and downward simulations [5, 10]. To verify a refinement the simulations use a retrieve relation which relates the concrete to abstract states and allow the comparison between the data types to be made on a step by step basis by comparing an abstract operation with its concrete counterpart. Versions of the simulation rules for Z are given in [9].

One approach to refining an abstract system is to *calculate* the concrete specification based upon the abstract specification, the concrete state space and a given retrieve relation which links the abstract and concrete states. The calculated concrete specification is then the most general refinement with respect to this retrieve relation, i.e., any other refinement will be a refinement of this calculated specification. It is useful then to be able to find the simplest means to calculate both upward and downward simulations of a given data type specification.

The principal work in this area is that of [5], where they consider refinement calculations in the context of total relations (partial operations are first totalised). However, in Z operations can be partial and our purpose is to derive equivalent results in a partial setting. This will be seen to generalise and simplify results in [7].

The structure of the paper is as follows. We discuss refinement in Z in Section 2, and in Section 3 we show how we can calculate refinements from a given retrieve relation. We simplify the result for downward simulation given in [7] and derive a result for upward simulations (which weren't considered in [7]). The final section makes some concluding remarks.

2 Refinement

In this section we discuss refinement in Z. We begin with an example followed by a discussion of the relational basis for the Z refinement rules, and it is this relational basis we use subsequently when deriving calculational methods of refinement. Throughout the paper we assume the reader is familiar with the Z notation.

Example:
In our example (adapted from [7]) the abstract specification consists of two sequences s and t (both initially empty). There are two operations: $push_A$ and pop_A. The $push_A$ operation has two inputs and pushes $m?$ into one of the sequences according to whether $i?$ is 1 or 0. The pop_A operation non-deterministically pops one of the sequences when either is non-empty, and outputs an error message if they are both empty. This operation is specified as the disjunction of two operations pop_{OkA} and pop_{ErrorA}. The specification is as follows.

$$\begin{array}{|l}
\multicolumn{1}{l}{Astate} \\ \hline
s, t : \operatorname{seq} \mathbb{N} \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{Ainit} \\ \hline
\Delta Astate \\ \hline
s' = t' = \langle\,\rangle \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{push_A} \\ \hline
\Delta Astate \\
m? : \mathbb{N} \\
i? : \{0, 1\} \\ \hline
(i? = 0 \land t' = t \frown \langle m? \rangle \land s' = s) \lor (i? = 1 \land s' = s \frown \langle m? \rangle \land t' = t) \\ \hline
\end{array}$$

$$
\begin{array}{|l}
\hline
pop_{OkA} \\
\Delta Astate \\
n! : \mathbb{N} \\
\hline
(t = t' \frown \langle n! \rangle \wedge s' = s) \vee (s = s' \frown \langle n! \rangle \wedge t' = t) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
pop_{ErrorA} \\
\Delta Astate \\
report! : REPORT \\
\hline
s = \langle \rangle \wedge t = \langle \rangle \wedge report! = \text{``}error - nothing\ to\ pop\text{''} \\
\hline
\end{array}
$$

$$pop_A \mathrel{\widehat{=}} pop_{OkA} \vee pop_{ErrorA}$$

In fact it is possible to refine this to a concrete specification whose state space consists of a single sequence u. That is, the two separate sequences were actually unnecessary in terms of the observable behaviour which consists of output values that are just some valid merge of the input streams. So the nondeterminism in the pop_{OkA} operation about which sequence is popped can be replaced by the nondeterminism of taking any valid merge of s and t in a single sequence. The concrete specification will have the following state space

$$
\begin{array}{|l}
\hline
Cstate \\
u : \mathrm{seq}\, \mathbb{N} \\
\hline
\end{array}
$$

together with an initialisation and operations $push_C$ and pop_C. Given such a complete concrete specification we would then have to verify that this is a refinement (in fact it would be a downward simulation) by using a retrieve relation R and show that it satisfies the following for the push and the pop operations.

Definition 1 *Let R be the retrieve relation between data types $(Astate, Ainit, \{AOp_i\})$ and $(Cstate, Cinit, \{COp_i\})$. R is a downwards simulation if the following hold for all operations.*

$$
\begin{array}{l}
\forall\, Cstate \bullet CInit \Rightarrow (\exists\, Astate \bullet AInit \wedge R) \\
\forall\, Astate;\ Cstate \bullet \mathrm{pre}\, AOp_i \wedge R \Rightarrow \mathrm{pre}\, COp_i \\
\forall\, Astate;\ Cstate;\ Cstate' \bullet \mathrm{pre}\, AOp_i \wedge COp_i \wedge R \Rightarrow \exists\, Astate' \bullet R' \wedge AOp_i
\end{array}
$$

The retrieve relation we would use here would be

$$
\begin{array}{|l}
\hline
R \\
Astate \\
Cstate \\
\hline
u_{merge}(s, t) \\
\hline
\end{array}
$$

where the predicate in the retrieve relation defines a merge of the two sequences s and t, and for example has the properties: $u_{merge}(s, \langle \rangle)$ iff $u = s$ and $u_{merge}(s, t)$ iff $u_{merge}(t, s)$.

An alternative approach to this aspect of software engineering is to move the emphasis from verification to calculation. That is instead of writing down the concrete operations and verifying they are refinements, it is possible to calculate the operations and initialisation. All that is needed is the description of the concrete state space and a retrieve relation which links the abstract to concrete. The result will be the most general refinement of the abstract specification with respect to the concrete state space and retrieve relation used.

There are clear advantages in moving effort from verification to calculation in terms of complexity and automation of the process - providing the calculations are simple enough. As mentioned above there are two forms of refinement: downward and upward simulations. For state based specifications there are known ways to calculate refinements that are downward simulations. The contribution of this paper is to simplify those calculations (Section 3.1) and derive analogous calculations for upward simulations (Section 3.2).

2.1 The relational basis for simulations

In this subsection we discuss the relational view of refinement and describe how it treats partiality, leading to the standard presentation of refinement in a language such as Z [8, 9]. In doing so we present a summary of results in [5, 4, 9] to which the reader is directed for more detailed explanation if necessary.

We shall need the following relational notation. $\fatsemi$ denotes relational composition, $\lhd$ is domain restriction, $\ntrianglerighteq$ is range subtraction, $\ntrianglelefteq$ is domain subtraction, and $\overline{X}$ is the complement of X. If S is a relation, then $A \lhd S = \{(x, y) \mid (x, y) \in S \land x \in A\}$, $A \ntrianglelefteq S = \{(x, y) \mid (x, y) \in S \land x \notin A\}$, and $S \ntrianglerighteq B = \{(x, y) \mid (x, y) \in S \land y \notin B\}$.

The underlying model of a state based system is a relational model, where the components (e.g. operation schemas in Z) of an abstract data type (ADT) are relations. An ADT is a quadruple $\mathcal{A} = (Astate, ai, \{aop_i\}_{i \in I}, af)$ which acts on a global state space G such that

- $Astate$ is the space of values;
- $ai \in G \leftrightarrow Astate$ is an initialisation;
- $af \in Astate \leftrightarrow G$ is a finalisation;
- aop_i are operations in $Astate \leftrightarrow Astate$.

Assuming for the moment that all the relations are total, a program P is then a sequence of operations upon a data type beginning with an initialisation and ending with a finalisation, e.g.

$$P(\mathcal{A}) = ai \fatsemi aop_1 \fatsemi aop_2 \fatsemi af$$

We can now consider refinement between two ADTs. It is assumed that the abstract and concrete data types have the same global state space G and that the indexing sets for the operations coincide (i.e., every abstract operation has a concrete counterpart and vice versa). Refinement is then defined as being the reduction of non-determinism, i.e. a data type $\mathcal{C}$ refines a data type $\mathcal{A}$ if, for every program P, $P(\mathcal{C}) \subseteq P(\mathcal{A})$.

This definition involves quantification over all programs, and in order to verify such refinements, *simulations* are used which consider values produced at each step of a program's execution. Simulations are thus the means to make the verification of a refinement feasible. In order to consider values produced at each step we need a relation r between the two state spaces $Astate$ and $Cstate$; this relation is known as the *retrieve* relation. The schema R in the example above is an example of a retrieve relation.

So far we have assumed that all the relations in a specification are total. However, in practice this is not the case (e.g. the operation pop_A above is partial), and the meaning of an operation

ρ specified as a partial relation is that ρ behaves as specified when used within its precondition (domain), and outside its precondition, anything may happen.

Therefore in order to apply refinement to such specifications we have to totalise their partial relations, i.e. in the semantics we add a distinguished element $\perp$ to the state space, denoting undefinedness, and $X^{\perp}$ denotes the augmented version of X. Thus if ρ is a partial relation between X and Y, we add the following sets of pairs to ρ

$$\{x : X^{\perp}, y : Y^{\perp} \mid x \notin \operatorname{dom}\rho \bullet x \mapsto y\}$$

and call this new (total) relation $\overset{\bullet}{\rho}$.

It is worth noting that this interpretation of the meaning of a partial relation differs between specification languages. For example, in Object-Z [2] outside a partial relation's precondition nothing may happen (i.e. preconditions denote guards). Different totalisations can be used to model these different interpretations. Some languages, such as B, have constructs which enable both interpretations to be specified.

The final requirement that we make is that the retrieve relation be strict, i.e., that r propagates undefinedness and we ensure this by considering the lifted form of $r \in X \leftrightarrow Y$:

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

With this in place we can consider the two types of step by step comparisons possible: downwards simulation and upwards simulation [9]. Their usefulness lies in the fact that they are sound and jointly complete [5].

A downward simulation is a relation r from $Astate$ to $Cstate$ such that

$$\begin{array}{ll} \overset{\bullet}{ci} \subseteq \overset{\bullet}{ai} \mathbin{⨾} \overset{\circ}{r} & \\ \overset{\circ}{r} \mathbin{⨾} \overset{\bullet}{cf} \subseteq \overset{\bullet}{af} & \\ \overset{\circ}{r} \mathbin{⨾} \overset{\bullet}{cop_i} \subseteq \overset{\bullet}{aop_i} \mathbin{⨾} \overset{\circ}{r} & \quad \text{for each index } i \in I \end{array}$$

An upward simulation is a relation l from $Cstate$ to $Astate$ such that

$$\begin{array}{ll} \overset{\bullet}{ci} \mathbin{⨾} \overset{\circ}{l} \subseteq \overset{\bullet}{ai} & \\ \overset{\bullet}{cf} \subseteq \overset{\circ}{l} \mathbin{⨾} \overset{\bullet}{af} & \\ \overset{\bullet}{cop_i} \mathbin{⨾} \overset{\circ}{l} \subseteq \overset{\circ}{l} \mathbin{⨾} \overset{\bullet}{aop_i} & \quad \text{for each index } i \in I \end{array}$$

These simulation rules are defined in terms of augmented relations. We can extract the underlying rules for the original partial relations as follows. For example, for a downwards simulation the above definition is equivalent to the following conditions

$$\begin{array}{l} ci \subseteq ai \mathbin{⨾} r \\ r \mathbin{⨾} cf \subseteq af \\ (\operatorname{dom} aop \lhd r \mathbin{⨾} cop) \subseteq aop \mathbin{⨾} r \\ \operatorname{ran}((\operatorname{dom} aop) \lhd r) \subseteq \operatorname{dom} cop \end{array}$$

The last two conditions mean that: the effect of cop must be consistent with that of aop; and, the operation cop is defined for every value that can be reached from the domain of aop using r.

We can also extract the underlying conditions in the definition of an upwards simulation, to find (see for example [9]) that they equivalent to the following conditions

$$
\begin{array}{l}
ci \fatsemi l \subseteq ai \\
cf \subseteq l \fatsemi af \\
\mathrm{dom}(l \mathbin{⩥} (\mathrm{dom}\, aop)) \mathbin{⩤} cop \fatsemi l \subseteq l \fatsemi aop \\
\overline{\mathrm{dom}\, cop} \subseteq \mathrm{dom}(l \mathbin{⩥} (\mathrm{dom}\, aop))
\end{array}
$$

The last two conditions mean: the effect of *cop* must be consistent with that of *aop*; and the set of values for which *cop* is not defined must be a subset of those for which *aop* is not defined.

These relational rules can now be used in a particular specification notation. For example, we can transform the rules from their relational setting to simulation rules for Z specifications by writing them in the Z schema calculus. The presentation of the downward simulation conditions in Z were given above, the upward simulation conditions are similar (see [9] for details). In Z (and VDM etc) we lose all explicit references to finalisation for reasons given in [9, 3].

3 Calculating refinements

In this section we consider the calculational aspects of refinement, and we develop rules for both upward and downward simulations. To do so we work in the relational setting, giving the results in Z as corollaries. Suppose we are given a specification of an abstract data type $\mathcal{A} = (Astate, ai, \{aop_i\}_{i \in I}, af)$, a concrete state space *Cstate* together with a retrieve relation r between *Astate* and *Cstate*. It is possible to calculate the most general refinement of $\mathcal{A}$, that is calculate the initialisation, finalisation and concrete operations.

As noted in [5], the calculations can be found by considering the most general solutions to the simulation requirements in the definitions given above. Therefore, the most general (i.e. weakest) solution for a downward simulation is given by:

$$
\begin{array}{ll}
\overset{\bullet}{ci} = \overset{\bullet}{ai} \fatsemi \overset{\circ}{r} & \\
\overset{\circ}{r} \fatsemi \overset{\bullet}{cf} = \overset{\bullet}{af} & \\
\overset{\circ}{r} \fatsemi \overset{\bullet}{cop}_i = \overset{\bullet}{aop}_i \fatsemi \overset{\circ}{r} & \text{for each index } i \in I
\end{array}
$$

which have explicit solutions (see [5]):

$$
\begin{array}{ll}
\overset{\bullet}{ci} = \overset{\bullet}{ai} \fatsemi \overset{\circ}{r} & \\
\overset{\bullet}{cf} = \overset{\bullet}{af} \;/\; \overset{\circ}{r} & \\
\overset{\bullet}{cop}_i = (\overset{\bullet}{aop}_i \fatsemi \overset{\circ}{r}) / \overset{\circ}{r} & \text{for each index } i \in I
\end{array}
$$

where $X/R = \overline{(R^{-1} \fatsemi \overline{X})}$.

Similarly, for an upward simulation the weakest solution is given by:

$$
\begin{array}{ll}
\overset{\bullet}{ci} = \overset{\circ}{l} \setminus \overset{\bullet}{ai} & \\
\overset{\bullet}{cf} = \overset{\circ}{l} \fatsemi \overset{\bullet}{af} & \\
\overset{\bullet}{cop}_i = \overset{\circ}{l} \setminus (\overset{\circ}{l} \fatsemi \overset{\bullet}{aop}_i) & \text{for each index } i \in I
\end{array}
$$

where $L \setminus X = \overline{(\overline{X} \fatsemi L^{-1})}$. We now consider how to simplify these conditions and to extract the calculation on the underlying partial relations. We begin with downward simulations.

3.1 Downward simulations

In this section the main result (Theorem 2) is the simplification of existing calculational methods for downward simulations. Extracting the calculations for the initialisation and finalisation is easy since we know that $\overset{\bullet}{ci} \subseteq \overset{\bullet}{ai} \mathbin{;} \overset{\circ}{r}$ iff $ci \subseteq ai \mathbin{;} r$ etc. Therefore the weakest concrete initialisation and finalisation are given by

$$
\begin{aligned}
ci &= ai \mathbin{;} r \\
cf &= af / r
\end{aligned}
$$

To calculate the concrete operations we note that the calculation $\overset{\bullet}{cop} = (\overset{\bullet}{aop} \mathbin{;} \overset{\circ}{r}) / \overset{\circ}{r}$ can be rewritten as two conditions: $(\mathrm{dom}\ aop \lhd r \mathbin{;} cop) = aop \mathbin{;} r$ and $\mathrm{ran}((\mathrm{dom}\ aop) \lhd r) = \mathrm{dom}\ cop$. Therefore cop is given by the weakest solution which is:

$$cop = \mathrm{ran}(\mathrm{dom}\ aop \lhd r) \lhd ((aop \mathbin{;} r)/(\mathrm{dom}\ aop \lhd r))$$

However, for a partial relation we also need to check applicability, and only if this concrete operation satisfies the applicability condition does a downward simulation exist. We summarise this in the following theorem.

Theorem 1 *The weakest data type that is a downward simulation of $\mathcal{A}$ with respect to r is given by*

$$
\begin{aligned}
ci &= ai \mathbin{;} r \\
cf &= af / r \\
cop &= ran(dom\ aop \lhd r) \lhd ((aop \mathbin{;} r)/(dom\ aop \lhd r))
\end{aligned}
$$

whenever $ran((dom\ aop) \lhd r) \subseteq dom\ cop$. If the latter does not hold then no downward simulation is possible for this $\mathcal{A}$ and r.

This theorem concurs with the results in [7] which were given in terms of the Z schema calculus. [7] also comments that in the case of r^{-1} defining a (partial) *surjective function* from *Cstate* to *Astate*, then the calculation simplifies to $cop = r^{-1} \mathbin{;} aop \mathbin{;} r$, and that in this case it is not necessary to check that $\mathrm{ran}((\mathrm{dom}\ aop) \lhd r) \subseteq \mathrm{dom}\ cop$. We show now that we can relax this hypothesis. In particular, it is not necessary that r^{-1} is surjective, and in addition r^{-1} does not have to be completely functional, it is sufficient that it is functional on a restricted domain.

Theorem 2 *Let $cop = ran(dom\ aop \lhd r) \lhd ((aop \mathbin{;} r)/(dom\ aop \lhd r))$. Then $cop \subseteq r^{-1} \mathbin{;} aop \mathbin{;} r$, and if $ran((dom\ aop) \lhd r) \subseteq dom\ cop$ and $dom\ aop \lhd r \mathbin{;} r^{-1} \subseteq id$ then $cop = r^{-1} \mathbin{;} aop \mathbin{;} r$.*

Proof

We first of all show that $cop \subseteq r^{-1} \mathbin{;} aop \mathbin{;} r$. Let $(a, b) \in cop$. Then $(\exists s \bullet (s, a) \in (\mathrm{dom}\ aop \lhd r)) \wedge (\forall s \bullet (s, a) \notin (\mathrm{dom}\ aop \lhd r) \vee (s, b) \in (aop \mathbin{;} r))$. Hence there exists an s such that $(s, a) \in (\mathrm{dom}\ aop \lhd r)$ and $(s, b) \in (aop \mathbin{;} r)$, and therefore $(a, b) \in r^{-1} \mathbin{;} aop \mathbin{;} r$.

Next we show that if $\mathrm{ran}((\mathrm{dom}\ aop) \lhd r) \subseteq \mathrm{dom}\ cop$ then $r^{-1} \mathbin{;} aop \mathbin{;} r \subseteq cop$. To do so suppose that $(a, b) \in r^{-1} \mathbin{;} aop \mathbin{;} r$, then there exists s such that $(s, a) \in (\mathrm{dom}\ aop \lhd r)$ and $(s, b) \in (aop \mathbin{;} r)$. We have to show that $\forall u \bullet (u, a) \notin (\mathrm{dom}\ aop \lhd r) \vee (u, b) \in (aop \mathbin{;} r)$. Consider any u with $(u, a) \in (\mathrm{dom}\ aop \lhd r)$, it suffices to show that $(u, b) \in (aop \mathbin{;} r)$.

Since $(u, a) \in (\mathrm{dom}\ aop \lhd r)$ and $(a, b) \in r^{-1} \mathbin{;} aop \mathbin{;} r$, we find that $(u, b) \in \mathrm{dom}\ aop \lhd r \mathbin{;} r^{-1} \mathbin{;} aop \mathbin{;} r \subseteq aop \mathbin{;} r$ since $\mathrm{dom}\ aop \lhd r \mathbin{;} r^{-1} \subseteq id$.

Therefore $\forall u \bullet (u,a) \notin (\mathrm{dom}\, aop \lhd r) \lor (u,b) \in (aop \mathbin{;} r)$. We also know that $\exists s \bullet (s,a) \in (\mathrm{dom}\, aop \lhd r)$ and $(s,b) \in (aop \mathbin{;} r)$. Thus by the definition of cop, $(a,b) \in cop$. □

Note also that in the case that r defines a function (not necessarily total or surjective) from *Cstate* to *Astate*, then it is not necessary to check that applicability holds.

The consequences of this theorem are the following. For the simpler calculation $cop = r^{-1} \mathbin{;} aop \mathbin{;} r$ to be used

- it is not necessary that r^{-1} is surjective, i.e. not every abstract state needs to be linked to a concrete state;
- it is not necessary that r^{-1} is a function, it only has to be functional on the smaller set $\mathrm{ran}(\mathrm{dom}\, aop \lhd r)$.

As we shall see in a moment these are sufficient, but not necessary, conditions, e.g. there are occasions where the simplified calculation can still be used even when r^{-1} is not functional at all. It is also easy to construct examples where the simplified calculation cannot be used when the necessary conditions on r fail.

We can describe these results in the Z schema calculus. To do so let R be the retrieve relation, let *Astate* be the abstract state space, *Ainit* the abstract initialisation, and let every abstract operation AOp have a concrete counterpart COp.

Corollary 1 *Given an abstract specification, a concrete state space and a retrieve relation R, the most general downward simulation can be calculated as:*

$$Cinit \mathrel{\widehat{=}} \exists\, Astate \bullet Ainit \land R$$
$$COp \mathrel{\widehat{=}} \exists\, Astate;\ Astate' \bullet (R \land AOp \land R')$$

whenever a downward simulation exists (which is guaranteed to do so when R is functional from concrete to abstract) and whenever R is a function from Cstate to Astate on $\mathit{ran}(\mathit{dom}\, AOp \lhd R)$.

Example:
We can apply this result to our example given in Section 2. This example is interesting because the retrieve relation is not functional: for every u there are many choices of s and t with $u_{merge}(s,t)$. In [7] Josephs thus uses complex calculations to produce the concrete operations, for example, $push_C$ is calculated by

$$push_C \mathrel{\widehat{=}} (\exists\, s,t : \mathrm{seq}\,\mathbb{N} \bullet \mathrm{pre}\, push_A \land R) \land (\forall\, s,t \bullet \mathrm{pre}\, push_A \land R \Rightarrow \exists\, s',t' \bullet push_A \land R')$$

The retrieve relation R is not functional on $\mathrm{ran}(\mathrm{dom}\, push_A \lhd R)$ which is the whole of *Cstate* since $push_A$ is total. So we cannot automatically use the simple calculation $push_C \mathrel{\widehat{=}} \exists\, t,s,t',s' : \mathrm{seq}\,\mathbb{N} \bullet (R \land push_A \land R')$.

However, the retrieve relation is functional on $\mathrm{ran}(\mathrm{dom}\, pop_{ErrorA} \lhd R)$, i.e. R links $u = \langle\,\rangle$ to only one abstract state (namely when both s and t are also empty). Therefore the simple calculation

$$pop_{ErrorC} \mathrel{\widehat{=}} \exists\, t,s,t',s' : \mathrm{seq}\,\mathbb{N} \bullet (R \land pop_{ErrorA} \land R')$$

can be used. This evaluates to

$$\begin{array}{|l}
\multicolumn{1}{l}{pop_{ErrorC}} \\
\hline
\Delta Cstate \\
report! : REPORT \\
\hline
u = \langle\,\rangle \land report! = \text{“}error - nothing\ to\ pop\text{''} \\
\hline
\end{array}$$

Incidentally, this example shows that the condition of functionality is not necessary since, for example, the calculation

$$pop_{OkC} \mathrel{\widehat=} \exists\, t, s, t', s' : \operatorname{seq} \mathbb{N} \bullet (R \land pop_{OkA} \land R')$$

evaluates to the same schema as the more complex calculation used in [7], namely

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad pop_{OkC} \quad}} \\
\Delta \mathit{Cstate} \\
n! : \mathbb{N} \\
\hline
\exists\, t, s \bullet u_{merge}(t \frown \langle n! \rangle, s) \land u'_{merge}(t, s) \\
\hline
\end{array}$$

Note that one obvious further refinement of this would be the operation

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad pop_{OkC} \quad}} \\
\Delta \mathit{Cstate} \\
n! : \mathbb{N} \\
\hline
t = t' \frown \langle n! \rangle \\
\hline
\end{array}$$

$\square$

3.2 Upward simulations

Turning to the case of upward simulations, we can produce analogous results, and again it is easy to extract the calculations for the initialisation and finalisation. They are given by:

$$\begin{aligned} ci &= l \setminus ai \\ cf &= l \mathbin{⨾} af \end{aligned}$$

To calculate the concrete operations the equation $\overset{\bullet}{cop} = \overset{\circ}{l} \setminus \overline{(\overset{\circ}{l} \mathbin{⨾} \overset{\bullet}{aop})}$ is equivalent (see [9]) to the two conditions $\mathrm{dom}(l \rhd (\mathrm{dom}\, aop)) \lhd cop \mathbin{⨾} l = l \mathbin{⨾} aop$ and $\overline{\mathrm{dom}\, cop} \subseteq \mathrm{dom}(l \rhd (\mathrm{dom}\, aop))$.

Therefore the weakest solution if one exists is given by $\mathrm{dom}(l \rhd (\mathrm{dom}\, aop)) \lhd cop = l \setminus (l \mathbin{⨾} aop)$, and hence

$$cop = \mathrm{dom}(l \rhd (\mathrm{dom}\, aop)) \lhd (l \setminus (l \mathbin{⨾} aop))$$

This will be a refinement whenever this cop satisfies the applicability condition $\overline{\mathrm{dom}\, cop} \subseteq \mathrm{dom}(l \rhd (\mathrm{dom}\, aop))$. If cop is total this condition is true since in this case $\overline{\mathrm{dom}\, cop} = \varnothing$. If the applicability condition fails then no upward simulation is possible for this $\mathcal{A}$ and l. We can summarise this as follows.

Theorem 3 *The weakest data type that is an upward simulation of $\mathcal{A}$ with respect to l is given by*

$$\begin{aligned} ci &= l \setminus ai \\ cf &= l \mathbin{⨾} af \\ cop &= dom(l \rhd (dom\, aop)) \lhd (l \setminus (l \mathbin{⨾} aop)) \end{aligned}$$

whenever $\overline{dom\, cop} \subseteq dom(l \rhd (dom\, aop))$. If the latter does not hold then no upward simulation is possible for this $\mathcal{A}$ and l.

We can now simplify this calculation in a fashion similar to that described for downward simulations. Note first that an upward simulation l must be total from concrete to abstract. This is due to the totality of a finalisation and the condition that $cf \subseteq l \,\S\, af$. The simplification of the calculation will then depend upon whether l is functional.

Theorem 4 *Let* $cop = dom(l \rhd (dom\ aop)) \lhd (l \setminus (l \,\S\, aop))$. *Then whenever* l *is a function from* $Cstate$ *to* $Astate$, $cop = l \,\S\, aop \,\S\, l^{-1}$.

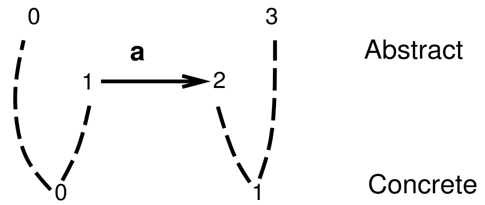
Proof
Let $(a, b) \in cop$. Then $a \notin \mathrm{dom}(l \rhd (\mathrm{dom}\ aop))$ and $(a, b) \in (l \setminus (l \,\S\, aop))$. Hence, $\forall \beta \bullet (a, \beta) \in l \Rightarrow \beta \in \mathrm{dom}\ aop$ and $\forall c \bullet (b, c) \in l \Rightarrow (a, c) \in (l \,\S\, aop)$. By the assumption of totality there exists at least one c with $(b, c) \in l$, and hence $(a, c) \in (l \,\S\, aop)$. Thus $(a, b) \in l \,\S\, aop \,\S\, l^{-1}$.

For the converse we need the assumption of functionality. Let $(a, b) \in l \,\S\, aop \,\S\, l^{-1}$. To show that $(a, b) \in cop$ we need to show that $a \notin \mathrm{dom}(l \rhd (\mathrm{dom}\ aop))$ and $(a, b) \in (l \setminus (l \,\S\, aop))$. For the former this amounts to showing that $\forall y \bullet (a, y) \notin l \vee y \in \mathrm{dom}\ aop$. However, since l is a function and $(a, b) \in l \,\S\, aop \,\S\, l^{-1}$ there is precisely one y with $(a, y) \in l$ and for this y we know that $y \in \mathrm{dom}\ aop$.

Showing that $(a, b) \in (l \setminus (l \,\S\, aop))$ amounts to showing that $\forall c \bullet (b, c) \in l \Rightarrow (a, c) \in (l \,\S\, aop)$, and again by the functionality of l this is easily seen to be true. Hence $(a, b) \in cop$. $\square$

Note that in fact the functionality of l can actually be weakened to requiring that $\mathrm{dom}\ aop \lhd l^{-1} \,\S\, l \subseteq id$ and $\mathrm{ran}\ aop \lhd l^{-1} \,\S\, l \subseteq id$.

Example:
The following example shows that the simplification does not always hold. The diagram depicts an abstract data type with state space $\{0, \ldots, 3\}$ and one operation $a = \{(1, 2)\}$. The concrete state space has just two points $\{0, 1\}$ and we are given a total relation l as our retrieve relation, where $l = \{(0, 0), (0, 1), (1, 2), (1, 3)\}$.



Then $cop = \mathrm{dom}(l \rhd (\mathrm{dom}\ aop)) \lhd (l \setminus (l \,\S\, aop)) = \varnothing$, however, $l \,\S\, aop \,\S\, l^{-1} = \{(0, 1)\}$. Therefore $l \,\S\, aop \,\S\, l^{-1}$ is not the most general upward simulation with this retrieve relation.

Finally, let us describe these results in the Z schema calculus. Again let R be the retrieve relation, let $Astate$ be the abstract state space, $Ainit$ the abstract initialisation, and let every abstract operation AOp have a concrete counterpart COp.

Corollary 2 *Given an abstract specification, a concrete state space and a retrieve relation* R*, the most general upward simulation can be calculated as:*

$$Cinit \mathrel{\widehat=} \forall Astate \bullet (R \Rightarrow Ainit)$$
$$COp \mathrel{\widehat=} \forall Astate \bullet (R \Rightarrow pre\ AOp) \wedge (\forall Astate' \bullet (R' \Rightarrow \exists Astate \bullet R \wedge AOp))$$

whenever an upward simulation exists. In the case when R *is functional from concrete to abstract,* COp *is given by*

$$COp \mathrel{\widehat=} \exists Astate;\ Astate' \bullet (R \wedge AOp \wedge R')$$

and in this case there is no need to check applicability.

4 Conclusions

In this note we have considered the calculation of refinements in state-based systems and in particular the Z specification language. We have simplified the existing result for calculations of downward simulations, and illustrated via an example how such calculations are carried out to produce the operations in a concrete specification. We have also derived a similar result for upward simulations. A small example illustrated that we cannot in general simplify the calculations of upward simulations.

References

[1] J. R. Abrial. *The B-Book: Assigning programs to meanings.* CUP, 1996.

[2] R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. *Computer Standards and Interfaces*, 17:511–533, September 1995.

[3] Kai Engelhardt and W-P de Roever. *Model-Oriented Data Refinement.* To appear.

[4] He Jifeng, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In B. Robinet and R. Wilhelm, editors, *Proc. ESOP 86*, volume 213, pages 187–196. Springer-Verlag, 1986.

[5] He Jifeng and C.A.R. Hoare. Prespecification and data refinement. In *Data Refinement in a Categorical Setting*, Technical Monograph, number PRG-90. Oxford University Computing Laboratory, November 1990.

[6] C. B. Jones. *Systematic Software Development using VDM.* Prentice Hall, 1989.

[7] M. B. Josephs. The data refinement calculator for Z specifications. *Information Processing Letters*, 27:29–33, February 1988.

[8] J. M. Spivey. *The Z notation: A reference manual.* Prentice Hall, 1989.

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

[10] J. C. P. Woodcock and C. C. Morgan. Refinement of state-based concurrent systems. In D. Bjorner, C. A. R. Hoare, and H. Langmaack, editors, *VDM '90 VDM and Z - Formal Methods in Software Development*, LNCS 428, pages 340–351, Kiel, FRG, April 1990. Springer-Verlag.