A single complete refinement rule for Z

Abstract

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 can be encapsulated into two simulation rules: downward and upward simulations. These simulations are known to be sound and jointly complete for boundedly-nondeterministic specifications. In this note we derive a single complete refinement method and show how it may be formulated in Z, this is achieved by using possibility mappings. The use of possibility mappings themselves is not new, our aim here is to reformulate them for use within the Z specification language.

Keywords: Refinement; State-based systems; Z.

1 Introduction

This paper concerns methods by which we can refine systems specified in state-based specification languages such as Z [14], B [2] and VDM [10]. 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*, which can be understood as (total or partial) relations on the underlying state space.

Data refinement is the standard method 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 [8, 15]. 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 [15]. These refinement rules are known to be sound and jointly complete, that is any upward or downward simulation is a valid refinement, and any valid refinement can be proved correct by application of appropriate upward and downward simulations [9, 16]. It is also well known that each simulation method is incomplete on its own, that is, using a retrieve relation that simply links concrete to abstract states it is not possible to derive a single complete simulation rule.

However, a single complete method can be derived by using either predicate transformers [6] or possibility mappings (instead of retrieve relations) [11] in its formulation. In [6] predicate transformers are used instead of relations to derive a single method of refinement, and it is shown how upward and downward simulations are special cases of their method, which is therefore complete.

Possibility mappings are functions from concrete states to sets of abstract states, and were originally proposed as a method of refinement for I/O automata in [11]. By using possibility mappings instead of a retrieve relation a single complete method of refinement can be derived. Soundness and completeness for possibility mappings for automata are discussed in [13]. The use of possibility mappings in context of transition systems is given in [7] where the resultant rule is called failure simulation and is in essence the same as the relational characterisation we derive below. Other complete refinement methods include Lamport and Abadi's history and prophesy variable approach [1], the relationship between this and possibility mappings is discussed in [13]. [12] provides an overview of simulation methods for untimed and timed automata which surveys the relationship between many of these approaches.

There are practical advantages for remaining with a relational approach however. One is that refinement methods can be formulated within a particular language, for example the simulation methods can be expressed within the Z schema calculus. Another is that relational methods are amenable to refinement by *calculation*, that is, concrete specifications can be calculated from the abstract specification together with the retrieve relation.

The modest contribution we seek to make here is the use of possibility mappings within a relational context. Such an approach is discussed briefly in [5], and our aim here is to derive a single complete method of refinement for partial relations and give its explicit formulation within the Z schema calculus. We also show how to calculate refinements using a relational context. In [3] we simplify existing calculational methods for downward simulations and derive similar results for upward simulations, illustrating their application in Z. We apply similar methods in this paper to the single refinement rule.

The structure of the paper is as follows. We discuss the underlying relational view of refinement in Section 2, and describe how it treats partiality. Section 3 derives the method and section 4 applies it to Z. Calculations of refinement are discussed in section 5 and we conclude in section 6.

2 Background on refinement

In this section we discuss the relational view of refinement which forms the basis for refinement in language such as Z [14, 15], and describe how it treats partiality. In doing so we present a summary of results in [8, 9, 15].

The underlying model of a state based system is a relational model, where the components 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 $Astate \leftrightarrow Astate$.

Notation:
We shall need the following relational notation. $\fatsemi$ denotes relational composition, $\lhd$ is domain restriction, $\rhd$ is range subtraction, $⩤$ is domain subtraction, and $\overline{X}$ is the complement of X. If S is a relation of type $X \leftrightarrow Y$ and $A \subseteq X$, $B \subseteq Y$, then the relational operators are defined by: $A \lhd S = \{(x, y) \mid (x, y) \in S \wedge x \in A\}$, $A ⩤ S = \{(x, y) \mid (x, y) \in S \wedge x \notin A\}$, and $S \rhd B = \{(x, y) \mid (x, y) \in S \wedge y \notin B\}$.

We shall also use the standard notation for the weakest post- and pre- specification [9]. These are defined by $X/R = \overline{(R^{-1} \mathbin{;} \overline{X})}$ and $L \setminus X = \overline{(\overline{X} \mathbin{;} L^{-1})}$ respectively and are the approximate left and right inverses for composition (i.e. $R \mathbin{;} T \subseteq X$ iff $T \subseteq X/R$ etc).

Programs and Refinement
At this stage we consider all relations to be total. A program P is a sequence of operations upon a data type beginning with an initialisation and ending with a finalisation, e.g.

$$P(\mathcal{A}) = ai \mathbin{;} aop_1 \mathbin{;} aop_2 \mathbin{;} aop_3 \mathbin{;} af$$

To consider refinement we assume 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). We can now define refinement in the usual fashion as being the reduction of non-determinism when moving from abstract to concrete data type.

Definition 1 *A data type $\mathcal{C} = (Cstate, ci, \{cop_i\}_{i \in I}, cf)$ refines a data type $\mathcal{A}$ if, for every program P, $P(\mathcal{C}) \subseteq P(\mathcal{A})$.*

This definition of refinement 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 $Cstate$. Such a *retrieve* relation gives rise to two types of step by step comparisons: downwards simulation and upwards simulation [8, 9].

Definition 2 *A downward simulation (denoted $\sqsubseteq^{DS}$) is a relation r from Astate to Cstate such that*

$$\begin{array}{ll} ci \subseteq ai \mathbin{;} r & \\ r \mathbin{;} cf \subseteq af & \\ r \mathbin{;} cop_i \subseteq aop_i \mathbin{;} r & \quad \textit{for each index } i \in I \end{array}$$

Definition 3 *An upward simulation (denoted $\sqsubseteq^{US}$) is a relation l from Cstate to Astate such that*

$$\begin{array}{ll} ci \mathbin{;} l \subseteq ai & \\ cf \subseteq l \mathbin{;} af & \\ cop_i \mathbin{;} l \subseteq \ l \mathbin{;} aop_i & \quad \textit{for each index } i \in I \end{array}$$

These simulation relations are the basis for refinement methods in Z and other state based languages. However in the relational framework we have described so far the relations were assumed to be total. In Z (and VDM etc) operations (and the relations they represent) are not necessarily total, 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.

Partiality
In order to represent partial operations in our framework (and hence define refinement for partial operations) the relational theory is extended by totalising partial relations. To do this we add a distinguished element $\perp$ to the state space, denoting undefinedness, and the augmented version of X is denoted $X^{\perp}$. Then 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, following [15], call this new (total) relation $\overset{\bullet}{\rho}$.

Different specification languages have different interpretations for the meaning of a partial relations. For example, in Object-Z [4] outside a partial relation's precondition nothing may happen (i.e. preconditions denote guards). In order to model these different interpretations we use different totalisations. Some languages, such as B, have constructs which enable both interpretations to be specified.

It is also necessary to restrict ourselves to *finitary* abstract level data types and finitary retrieve relations. Recall that a non-empty subset of a data type is finitary if it is either finite or the whole type. A relation is finitary if the image of every element is finitary [9]. The restriction to finitary abstract level data types ensures that unbounded nondeterminism cannot be introduced into abstract level specifications, and is necessary to preserve soundness and completeness of the simulations.

We also require that the retrieve relation be strict, i.e., that r propagates undefinedness and we ensure this by considering the lifted form of $r \in X \leftrightarrow Y$:

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

The difference between the relational operators $\bullet$ and $\circ$ is the following. $\bullet$ makes a relation total by providing images for every element outside the domain, whereas $\circ$ merely propagates undefinedness by adding $\perp$ to the domain and mapping it to every element in the range. This is needed to ensure that the relational composition of relations which are undefined is also undefined, a property that is needed in a retrieve relation.

Refinement of partial operations
Refinement and simulations can now be applied to specifications involving partial operations by first totalising the relations that represent them and lifting the retrieve relation. Thus the requirements for a downwards simulation are:

$$\overset{\bullet}{ci} \subseteq \overset{\bullet}{ai} \fatsemi \overset{\circ}{r}$$
$$\overset{\circ}{r} \fatsemi \overset{\bullet}{cf} \subseteq \overset{\bullet}{af}$$
$$\overset{\circ}{r} \fatsemi \overset{\bullet}{cop_i} \subseteq \overset{\bullet}{aop_i} \fatsemi \overset{\circ}{r} \qquad \text{for each index } i \in I$$

and similarly for upward simulations. Simulations are sound and jointly complete in the following sense [9].

Theorem 1

- *If there is a downward simulation from Astate to Cstate, or an upward simulation from Cstate to Astate, then $\mathcal{C}$ refines $\mathcal{A}$.*
- *Every valid refinement can be verified as a sequence of downward and upward simulations.*

Abstract finalisations are also required to terminate weakly [5], i.e. it must be the case that $\perp$ is not in $\mathrm{ran}(Astate \lhd af)$. However, since finalisations are typically projections into the global state space this assumption is reasonable [6]. In particular, finalisations in Z are weakly terminating.

To derive a single complete simulation we will use the construction used in the joint completeness proof. The completeness result is the following: If data type $\mathcal{A}$ is refined by $\mathcal{C}$, then there is a data type $\mathcal{CA} = (CAstate, cai, \{caop_i\}_{i \in I}, caf)$ such that there is an upward simulation from $\mathcal{CA}$ to $\mathcal{A}$ and a downward simulation from $\mathcal{CA}$ to $\mathcal{C}$. The data type $\mathcal{CA}$ is given constructively and is equivalent to $\mathcal{A}$ (there are simulations both ways). It is also *canonical*, i.e. all operations and the initialisation are functions, so that all non-determinism present in $\mathcal{A}$ has been factored out.

3 A single simulation rule

To derive a single complete simulation rule, which we call powersimulation after [6], we use the construction discussed above as follows. Given a refinement between data types $\mathcal{A}$ and $\mathcal{C}$ we first totalise their partial relations then construct the data type $\mathcal{CA}$. Using the upward simulation between $\mathcal{A}$ and $\mathcal{CA}$ and the downward simulation between $\mathcal{CA}$ and $\mathcal{C}$ we can derive equivalent conditions between $\mathcal{A}$ and $\mathcal{C}$ (i.e. we eliminate $\mathcal{CA}$). Throughout the remainder of this paper let $\mathcal{A} = (Astate, ai, \{aop_i\}_{i \in I}, af)$ and $\mathcal{C} = (Cstate, ci, \{cop_i\}_{i \in I}, cf)$ be (partial) data types such that $\mathcal{C}$ refines $\mathcal{A}$.

The construction used in the joint completeness proof defines an intermediate data type $\mathcal{CA} = (CAstate, cai, \{caop_i\}_{i \in I}, caf)$ with $\mathcal{A} \sqsubseteq^{US} \mathcal{CA} \sqsubseteq^{DS} \mathcal{C}$.

From the data type $\mathcal{A}$ the construction begins by defining the concrete state space of $\mathcal{CA}$ as $CAstate = \mathbb{P}(Astate^{\perp})$. It then defines a relation l which will define an upward simulation, and is given by

$$\alpha(l)\beta \text{ iff } \beta \in \alpha$$

This relation is then used to define the relations $cai, \{caop_i\}_{i \in I}, caf$ in $\mathcal{CA}$ as follows. For the sake of readability let us fix $i \in I$, and let aop, cop and $caop$ be corresponding operations in the data types, where $\overset{\bullet}{aop}$: $Astate^{\perp} \leftrightarrow Astate^{\perp}$, $\overset{\bullet}{cop}$: $Cstate^{\perp} \leftrightarrow Cstate^{\perp}$ and $caop : CAstate \leftrightarrow CAstate$. Then $cai, caop, caf$ are defined as the weakest solutions to the following equations.

$$cai \,⨾\, l = \overset{\bullet}{ai}$$
$$caf = l \,⨾\, \overset{\bullet}{af}$$
$$caop \,⨾\, l = l \,⨾\, \overset{\bullet}{aop}$$

Because $\mathcal{A}$ was a partial data type we totalise its relations, however, having done this the relations in $\mathcal{CA}$ are by definition total, and therefore the defining equations appear as $cai \,⨾\, l = \overset{\bullet}{ai}$ etc. We can employ standard techniques to calculate these relations completely.

The construction also defines a downward simulation r between $\mathcal{CA}$ and $\mathcal{C}$. That is, the data types satisfy the following set of equations.

$$\overset{\bullet}{ci} \subseteq cai \,⨾\, r$$
$$r \,⨾\, \overset{\bullet}{cf} \subseteq caf$$
$$r \,⨾\, \overset{\bullet}{cop} \subseteq caop \,⨾\, r$$

These are in terms of the totalised relations. To make the conditions practical we need to derive equivalent conditions on the underlying partial relations. This we do now.

To extract the conditions on the underlying partial relations from their totalisations we define additional domain restriction and subtraction operators $\lhd_P$ and $⩤_P$. We also define a restriction $\underline{r} : \mathbb{P}\, Astate \leftrightarrow Cstate$ of the retrieve relation r to defined values. The definitions are as follows.

Definition 4

$$dom\, aop \lhd_P caop = \{(\alpha, \beta) \mid (\alpha, \beta) \in caop \wedge \alpha \subseteq dom\, aop\}$$
$$dom\, aop ⩤_P caop = \{(\alpha, \beta) \mid (\alpha, \beta) \in caop \wedge \alpha \not\subseteq dom\, aop\}$$
$$\underline{r} = (\mathbb{P}\, Astate) \lhd r \rhd Cstate$$

These are analogous to the standard domain restriction and subtraction operators $\lhd$ and $⩤$ given above, however, as our simulation construction involves powersets in the state space of $\mathcal{CA}$ we are interested in restricting to *subsets* as opposed to *elements* (e.g. as in $\lhd$) of dom aop.

We first of all consider the upward simulation equation $caop \mathbin{;} l = l \mathbin{;} \dot{aop}$.

Lemma 1 *$caop \mathbin{;} l \subseteq l \mathbin{;} \dot{aop}$ iff $dom\, aop \lhd_P caop \subseteq l \setminus (l \mathbin{;} aop)$*

Proof
To begin we note that

$$\begin{array}{rcl} caop \mathbin{;} l & = & l \mathbin{;} \dot{aop} \\ & = & l \mathbin{;} (aop \cup \overline{\mathrm{dom}\, aop^{\perp}} \times Astate^{\perp}) \\ & = & l \mathbin{;} aop \cup l \mathbin{;} (\overline{\mathrm{dom}\, aop^{\perp}} \times Astate^{\perp}) \end{array}$$

Therefore

$$\begin{array}{rcl} caop \mathbin{;} l \subseteq l \mathbin{;} \dot{aop} & \text{iff} & caop \mathbin{;} l \subseteq l \mathbin{;} aop \cup l \mathbin{;} (\overline{\mathrm{dom}\, aop^{\perp}} \times Astate^{\perp}) \\ & \text{iff} & (\mathrm{dom}\, aop \lhd_P caop) \mathbin{;} l \subseteq l \mathbin{;} aop \\ & \text{iff} & \mathrm{dom}\, aop \lhd_P caop \subseteq l \setminus (l \mathbin{;} aop) \end{array}$$

□

In a similar vein we consider the downward simulation equation $r \mathbin{;} \dot{cop} \subseteq caop \mathbin{;} r$.

Lemma 2 *$r \mathbin{;} \dot{cop} \subseteq caop \mathbin{;} r$ is equivalent to the following conditions:*

$dom\, aop \lhd_P (\underline{r} \mathbin{;} cop) \subseteq (dom\, aop \lhd_P caop) \mathbin{;} \underline{r}$
$ran(dom\, aop \lhd_P \underline{r}) \subseteq dom\, cop$

Proof
Suppose for the moment that r satisfies a condition (*), namely that

$\alpha(r)\perp \Rightarrow \perp \in \alpha$
$\perp \in \alpha \Rightarrow \alpha(r)\beta$ for all $\beta \in Cstate^{\perp}$

Then $r \mathbin{;} \dot{cop} \subseteq caop \mathbin{;} r$ if and only if

$$\begin{array}{rcll} r \mathbin{;} \dot{cop} & \subseteq & (\mathrm{dom}\, aop \lhd_P caop \mathbin{;} r) \cup (\mathrm{dom}\, aop ⩤_P caop \mathbin{;} r) & \\ & = & (\mathrm{dom}\, aop \lhd_P caop \mathbin{;} r) \cup \mathrm{dom}(\mathrm{dom}\, aop ⩤_P caop) \times Cstate^{\perp} & \text{by (*)} \end{array}$$

Therefore

$$\begin{array}{lll} r \mathbin{;} \dot{cop} \subseteq caop \mathbin{;} r & \text{iff} & \mathrm{dom}\, aop \lhd_P (r \mathbin{;} \dot{cop}) \subseteq (\mathrm{dom}\, aop \lhd_P caop) \mathbin{;} \underline{r} \\ & \text{iff} & \mathrm{dom}\, aop \lhd_P (\underline{r} \mathbin{;} \dot{cop}) \subseteq (\mathrm{dom}\, aop \lhd_P caop) \mathbin{;} \underline{r} \\ & \text{iff} & \mathrm{dom}\, aop \lhd_P (\underline{r} \mathbin{;} cop) \subseteq (\mathrm{dom}\, aop \lhd_P caop) \mathbin{;} \underline{r} \\ & \textit{and} & \mathrm{dom}\, aop \lhd_P (\underline{r} \mathbin{;} (\overline{\mathrm{dom}\, cop^{\perp}} \times Cstate^{\perp})) \subseteq (\mathrm{dom}\, aop \lhd_P caop) \mathbin{;} \underline{r} \end{array}$$

and the latter condition holds if and only if ran(dom $aop \lhd_P \underline{r}$) $\subseteq$ dom cop.

We have assumed that r satisfies (*), we have to justify this or show that any simulation relation r can in fact be replaced by one which does. It is easy to see that the latter option is always possible. For example, suppose that $\alpha(r)\perp$. Then we have $(\alpha, \beta) \in caop \fatsemi r$ for all $\beta \in Cstate^{\perp}$, since $r \fatsemi \overset{\bullet}{cop} \subseteq caop \fatsemi r$. Therefore we can assume without harm that $\perp \in \alpha$. The other condition can also shown to be safely assumed in a similar way. $\square$

Corollary 1 *The conditions in the upward and downward simulations $r \fatsemi \overset{\bullet}{cop} \subseteq caop \fatsemi r$ and $caop \fatsemi l = l \fatsemi \overset{\bullet}{aop}$ are equivalent to*

$$dom\, aop \lhd_P (\underline{r} \fatsemi cop) \subseteq (l \setminus (l \fatsemi aop)) \fatsemi \underline{r} \tag{1}$$

$$ran(dom\, aop \lhd_P \underline{r}) \subseteq dom\, cop \tag{2}$$

In a similar fashion we can consider the initialisation conditions $cai \fatsemi l = \overset{\bullet}{ai}$ and $\overset{\bullet}{ci} \subseteq cai \fatsemi r$. We find that, since the concrete and abstract initialisations are total, $\overset{\bullet}{ci} \subseteq cai \fatsemi r$ iff $ci \subseteq cai \fatsemi r$. Therefore the initialisation conditions in the upward and downward simulations are equivalent to

$$ci \subseteq (l \setminus ai) \fatsemi r \tag{3}$$

Similarly the finalisations $caf = l \fatsemi \overset{\bullet}{af}$ and $r \fatsemi \overset{\bullet}{cf} \subseteq caf$ are equivalent to

$$cf \subseteq (l \fatsemi af)/r \tag{4}$$

We can draw the conditions together to define a powersimulation as follows (we drop the underscore as we have no further use for it).

Definition 5 *A powersimulation between abstract data types $\mathcal{A}$ and $\mathcal{C}$ is a relation $r : \mathbb{P}\, Astate \leftrightarrow Cstate$ such that for all $i \in I$:*

$$\begin{array}{l} ci \subseteq (l \setminus ai) \fatsemi r \\ dom\, aop \lhd_P (r \fatsemi cop_i) \subseteq (l \setminus (l \fatsemi aop_i)) \fatsemi r \\ ran(dom\, aop_i \lhd_P r) \subseteq dom\, cop_i \\ cf \subseteq (l \fatsemi af)/r \end{array}$$

By its construction it is clear that every valid refinement can be verified as by one powersimulation, so powersimulations are complete. In addition every powersimulation is sound since a powersimulation is just a combination of an upward and downward simulation. It thus defines a single complete method for refinement.

4 Application to Z

A relational theory of refinement can be applied to particular specification languages. For example, the theory of upward and downward simulations for partial relations can be applied to Z specifications written in the Z schema calculus notation. This is achieved by considering each operation schema to be a partial relation on the state space, and the upward and downward simulations can then be restated in terms of schemas to produce the standard presentation of refinement in Z (see [15, 5]).

In order to derive a single complete simulation in Z we will apply the relational conditions we derived in the previous section to the Z schema calculus. To illustrate this let us consider a simple example written in Z. The abstract specification is given by

$$\begin{array}{|l} \multicolumn{1}{l}{Astate} \\ \hline x : 0..5 \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{Ainit} \\ \hline Astate' \\ \hline x' = 0 \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{Aone} \\ \hline \Delta Astate \\ \hline (x = 0 \land x' = 1) \lor (x = 1 \land x' = 0) \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{Atwo} \\ \hline \Delta Astate \\ \hline (x = 0 \land x' \in \{2, 3\}) \\ \hline \end{array} \qquad \begin{array}{|l} \multicolumn{1}{l}{Athree} \\ \hline \Delta Astate \\ \hline (x = 2 \land x' = 4) \lor (x = 3 \land x' = 5) \\ \hline \end{array}$$

The state space $Astate$, initialisation $Ainit$ and operations $Aone$ etc, are given as *schemas* which are used to structure the specification. A schema can be used as a type, predicate or declaration within a Z specification depending on the context. A simple relational semantics [15] allows us to view each specification as a relation, and therefore apply the refinement theory presented in the previous sections.

To see this we need to use the θ notation. If S is the name of a schema, then θS denotes the characteristic binding of components from S, where components are bound to the values of variables. Thus $\theta Astate$ denotes the characteristic binding of components from $Astate$, i.e. $\theta Astate = \langle\!| \; x \leadsto x \; |\!\rangle$. In order that schemas can be used as declarations, in a declaration the schema S is an abbreviation of $\{S \bullet \theta S\}$. For example, the schema $Astate$ is the appropriate set of bindings, i.e. $Astate = \{\langle\!| \; x \leadsto 0 \; |\!\rangle, \ldots, \langle\!| \; x \leadsto 5 \; |\!\rangle\}$. $\mathbb{P}\, Astate$ is then the powerset of this set of bindings. Likewise $\mathrm{pre}\, AOp$ describes a schema (the precondition of operation AOp), which can be considered as a set of bindings, and thus $\mathbb{P}(\mathrm{pre}\, Aone) = \{\varnothing, \{\langle\!| \; x \leadsto 0 \; |\!\rangle\}, \ldots, \{\langle\!| \; x \leadsto 0 \; |\!\rangle, \langle\!| \; x \leadsto 1 \; |\!\rangle\}\}$.

Suppose the data types $\mathcal{A}$ and $\mathcal{C}$ are described using schemas. To derive the powersimulation conditions our retrieve relation will be given as a relation $r : \mathbb{P}\, Astate \leftrightarrow Cstate$. Let us consider two operations AOp and COp and suppose that the initialisations in $\mathcal{A}$ and $\mathcal{C}$ are given by $Ainit$ and $Cinit$ respectively. Ignoring input and output for the moment the relations corresponding to these schemas are given by

$$\begin{array}{l} aop = \{AOp \bullet \theta Astate \mapsto \theta Astate'\} \\ cop = \{COp \bullet \theta Cstate \mapsto \theta Cstate'\} \\ ai = \{Ainit \bullet \theta Astate'\} \\ ci = \{Cinit \bullet \theta Cstate'\} \end{array}$$

(the last two are considered as relations where we hide the domain). We can now express each condition in the powersimulation in terms of schemas. For example, the initialisation condition becomes

$$\begin{array}{lll} ci \subseteq (l \setminus ai) \,⨾\, r & \text{iff} & \forall c : Cstate \bullet c \in ci \Rightarrow c \in (l \setminus ai) \,⨾\, r \\ & \text{iff} & \forall Cstate \bullet \theta Cstate \in ci \Rightarrow \theta Cstate \in (l \setminus ai) \,⨾\, r \\ & \text{iff} & \forall Cstate \bullet \theta Cstate \in \{Cinit \bullet \theta Cstate'\} \Rightarrow \exists \gamma : \mathbb{P}\, Astate \bullet r(\gamma) = \theta Cstate \land \gamma \in (l \setminus ai) \\ & \text{iff} & \forall Cstate' \bullet Cinit \Rightarrow \exists \gamma : \mathbb{P}\, Ainit \bullet r(\gamma) = \theta Cstate \end{array}$$

In a similar way the remaining conditions can be given in terms of schemas, and we have the following derivations:

$$\begin{array}{l} \mathrm{dom}\, aop \lhd_P (\underline{r} \,⨾\, cop) \subseteq (l \setminus (l \,⨾\, aop)) \,⨾\, \underline{r} \qquad \text{iff} \\ \forall \gamma : \mathbb{P}(\mathrm{pre}\, AOp);\; Cstate;\; Cstate' \bullet r(\gamma) = \theta Cstate \land COp \Rightarrow \\ \qquad\qquad \exists \gamma_2 : \mathbb{P}\, Astate \bullet r(\gamma_2) = \theta Cstate' \land \forall Astate' : \gamma_2 \bullet \exists Astate : \gamma \bullet AOp \end{array}$$

and

$$\text{ran}(\text{dom}\ aop \lhd_P \underline{r}) \subseteq \text{dom}\ cop \qquad \text{iff}$$
$$\forall\ Cstate \bullet \forall \gamma : \mathbb{P}(\text{pre}\ AOp) \bullet r(\gamma) = \theta Cstate \Rightarrow \text{pre}\ COp$$

Because finalisations in Z are projections into the global state space, the finalisation conditions are trivially met. We therefore do not need an explicit finalisation requirement for powersimulations.

Before we proceed with an example, a note is in order on the use of the schema notation within these expressions. Because expressions such as $Astate$, $\mathbb{P}\ Astate$, pre AOp and $\mathbb{P}(\text{pre}\ AOp)$ denote sets or powersets of bindings, the explicit conditions for powersimulations we have derived are formulated within the schema calculus itself. Schema decorations are ignored in determining the type of bindings (e.g. $\langle\!|\ x \leadsto 5\ |\!\rangle$ and $\langle\!|\ x' \leadsto 4\ |\!\rangle$ are of the same schema type, see [15]). This allows $r(\gamma) = \theta Cinit$ etc to have the obvious meaning.

In order to consider operations involving inputs and outputs, the standard approach is to embed them as components in the state space. This approach can be adopted here, and in fact we find that the form of the rules remain the same except we need to quantify over all possible inputs and outputs. We can now give the definition of powersimulation in Z.

Definition 6 *A powersimulation between abstract data types $\mathcal{A}$ and $\mathcal{C}$ written as schemas is a relation $r : \mathbb{P}\ Astate \leftrightarrow Cstate$ such that the following holds for every operation and initialisation.*

$$\forall\ Cstate' \bullet Cinit \Rightarrow \exists\, \gamma : \mathbb{P}\ Ainit \bullet r(\gamma) = \theta Cstate$$
$$\forall\, \gamma : \mathbb{P}(pre\ AOp);\ Cstate;\ Cstate' \bullet r(\gamma) = \theta Cstate \wedge COp \Rightarrow$$
$$\exists\, \gamma_2 : \mathbb{P}\ Astate \bullet r(\gamma_2) = \theta Cstate' \wedge \forall\ Astate' : \gamma_2 \bullet \exists\ Astate : \gamma \bullet AOp$$
$$\forall\ Cstate \bullet \forall\, \gamma : \mathbb{P}(pre\ AOp) \bullet r(\gamma) = \theta Cstate \Rightarrow pre\ COp$$

4.1 Example

We illustrate the refinement rules with a simple example written in Z. The abstract specification was given above, the concrete specification is given by

Cstate
$y : \{0, 2, 4, 5\}$

Cinit
$Cstate'$

$y' = 0$

Cone
$\Delta Cstate$

$y = y' = 0$

Ctwo
$\Delta Cstate$

$(y = 0 \wedge y' = 2)$

Cthree
$\Delta Cstate$

$(y = 2 \wedge y' \in \{4, 5\})$

The concrete specification is a refinement of the abstract, however, to verify this both downward and upward simulations are needed. With a powersimulation this can verified in one step. To do so we need to describe a retrieve relation between $\mathbb{P}\ Astate$ and $Cstate$, and in this example the following will suffice

$r : \mathbb{P}\ Astate \leftrightarrow Cstate$

$$\begin{aligned} r = \{ & \{\langle\!|\ x \leadsto 0\ |\!\rangle, \langle\!|\ x \leadsto 1\ |\!\rangle\} \mapsto \langle\!|\ y \leadsto 0\ |\!\rangle, \\ & \{\langle\!|\ x \leadsto 0\ |\!\rangle\} \mapsto \langle\!|\ y \leadsto 0\ |\!\rangle, \\ & \{\langle\!|\ x \leadsto 1\ |\!\rangle\} \mapsto \langle\!|\ y \leadsto 0\ |\!\rangle, \\ & \{\langle\!|\ x \leadsto 4\ |\!\rangle, \langle\!|\ x \leadsto 5\ |\!\rangle\} \mapsto \langle\!|\ y \leadsto 4\ |\!\rangle, \\ & \{\langle\!|\ x \leadsto 4\ |\!\rangle, \langle\!|\ x \leadsto 5\ |\!\rangle\} \mapsto \langle\!|\ y \leadsto 5\ |\!\rangle\} \end{aligned}$$

we now have to prove that the powersimulation conditions are satisfied. That is

$$
\begin{array}{l}
\forall\, Cstate' \bullet Cinit \Rightarrow \exists\, \gamma : \mathbb{P}\, Ainit \bullet r(\gamma) = \theta\, Cstate \\
\forall\, \gamma : \mathbb{P}(\mathrm{pre}\, AOp);\ Cstate;\ Cstate' \bullet r(\gamma) = \theta\, Cstate \wedge COp \Rightarrow \\
\qquad\qquad\qquad \exists\, \gamma_2 : \mathbb{P}\, Astate \bullet r(\gamma_2) = \theta\, Cstate' \wedge \forall\, Astate' : \gamma_2 \bullet \exists\, Astate : \gamma \bullet AOp \\
\forall\, Cstate \bullet \forall\, \gamma : \mathbb{P}(\mathrm{pre}\, AOp) \bullet r(\gamma) = \theta\, Cstate \Rightarrow \mathrm{pre}\, COp
\end{array}
$$

for every operation and initialisation. Proving the initialisation condition amounts to showing that

$$\forall\, y' : \{0, 2, 4, 5\} \bullet (y' = 0) \Rightarrow \exists\, \gamma : \mathbb{P}\{\varnothing, \{\langle\!|\, x' \leadsto 0 \,|\!\rangle\}\} \bullet r(\gamma) = \langle\!|\, y' \leadsto 0 \,|\!\rangle$$

which is clearly true. Similarly showing that

$$\forall\, Cstate \bullet \forall\, \gamma : \mathbb{P}(\mathrm{pre}\, Aone) \bullet r(\gamma) = \theta\, Cstate \Rightarrow \mathrm{pre}\, Cone$$

amounts to proving that

$$\forall\, y : \{0, 2, 4, 5\} \bullet \forall\, \gamma : \{\varnothing, \{\langle\!|\, x \leadsto 0 \,|\!\rangle\}, \ldots, \{\langle\!|\, x \leadsto 0 \,|\!\rangle, \langle\!|\, x \leadsto 1 \,|\!\rangle\}\} \bullet r(\gamma) = \theta\, Cstate \Rightarrow (y = 0)$$

This and the remaining conditions are easily shown to be true. $\square$

5 Calculating refinements

In this section we consider the calculational aspects of powersimulations. 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 $\mathbb{P}\, Astate$ and $Cstate$. It is possible to calculate the most general powersimulation of $\mathcal{A}$.

We first work in the relational setting and then give the corresponding results in Z. Extracting the calculations for the initialisation and finalisation are easy and the weakest concrete initialisation and finalisation are given by

$$
\begin{array}{l}
ci = (l \setminus ai) \mathbin{;} r \\
cf = (l \mathbin{;} af)/r
\end{array}
$$

Similarly the weakest solution of an operations aop is

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

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

Theorem 2 *The weakest data type that is a powersimulation of $\mathcal{A}$ with respect to r is given by*

$$
\begin{array}{l}
ci = (l \setminus ai) \mathbin{;} r \\
cf = (l \mathbin{;} af)/r \\
cop = ran(dom\, aop \lhd_P r) \lhd (((l \setminus (l \mathbin{;} aop)) \mathbin{;} r)/(dom\, aop \lhd_P r))
\end{array}
$$

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

Under certain circumstances a simplification is possible. In particular if the domain of r are singletons then the calculation simplifies to $cop = r^{-1} \fatsemi (l \setminus (l \fatsemi aop)) \fatsemi r$, and that in this case it is not necessary to check that $\mathrm{ran}((\mathrm{dom}\, aop) \lhd_P r) \subseteq \mathrm{dom}\, cop$.

Proposition 1 *Let* $cop = ran(dom\, aop \lhd_P r) \lhd (((l \setminus (l \fatsemi aop)) \fatsemi r)/(dom\, aop \lhd_P r))$. *Then* $cop \subseteq r^{-1} \fatsemi (l \setminus (l \fatsemi aop)) \fatsemi r$. *Furthermore if* $ran((dom\, aop) \lhd_P r) \subseteq dom\, cop$ *and the domain of* r *consists of singleton sets then* $cop = r^{-1} \fatsemi (l \setminus (l \fatsemi aop)) \fatsemi r$.

Proof
Let $(a, b) \in cop$. Then $(\exists\, s \bullet (s, a) \in (\mathrm{dom}\, aop \lhd_P r)) \wedge (\forall\, s \bullet (s, a) \notin (\mathrm{dom}\, aop \lhd_P r) \vee (s, b) \in ((l \setminus (l \fatsemi aop)) \fatsemi r))$. Hence there exists an s such that $(s, a) \in (\mathrm{dom}\, aop \lhd_P r)$ and $(s, b) \in ((l \setminus (l \fatsemi aop)) \fatsemi r)$, and therefore $(a, b) \in r^{-1} \fatsemi (l \setminus (l \fatsemi aop)) \fatsemi r$.

Conversely, let $(a, b) \in r^{-1} \fatsemi (l \setminus (l \fatsemi aop)) \fatsemi r$, and assume the domain of r are singletons and that $\mathrm{ran}((\mathrm{dom}\, aop) \lhd_P r) \subseteq \mathrm{dom}\, cop$. Then there exists s_1 such that $(s_1, a) \in r$ and $(s_1, b) \in ((l \setminus (l \fatsemi aop)) \fatsemi r)$. The latter is equivalent to $\exists\, s_2 \bullet (s_2, b) \in r \wedge \forall\, t \bullet t \notin s_2 \vee \exists\, u \bullet u \in s_1 \wedge (u, t) \in aop$. The condition on the domain of r implies that $s_1 \subseteq \mathrm{dom}\, aop$, and therefore that $(a, b) \in cop$. $\Box$

Note that in fact the simplification of cop to $cop = r^{-1} \fatsemi (l \setminus (l \fatsemi aop)) \fatsemi r$ only requires that $\forall\, s \in \mathrm{dom}\, r \bullet (s \cap \mathrm{dom}\, aop = \varnothing) \vee (s \subseteq \mathrm{dom}\, aop)$ for every abstract operation.

We can now describe these results in the Z schema calculus.

Corollary 2 *Given an abstract specification, a concrete state space and a retrieve relation* r *between the abstract and concrete state spaces, the most general powersimulation can be calculated as:*

$$\begin{aligned} &Cinit \mathrel{\widehat=} \exists\, \gamma : \mathbb{P}\, Ainit \bullet r(\gamma) = \theta\, Cstate \\ &COp \mathrel{\widehat=} \exists\, \gamma_1, \gamma_2 : \mathbb{P}\, Astate \bullet r(\gamma_1) = \theta\, Cstate \wedge r(\gamma_2) = \theta\, Cstate' \wedge \gamma_1 \subseteq pre\, AOp \wedge \\ &\qquad\qquad \forall\, Astate' : \gamma_2 \bullet \exists\, Astate : \gamma_1 \bullet AOp \end{aligned}$$

whenever a powersimulation exists. The simplification of COp, when appropriate, is given by

$$COp \mathrel{\widehat=} \exists\, \gamma_1, \gamma_2 : \mathbb{P}\, Astate \bullet r(\gamma_1) = \theta\, Cstate \wedge r(\gamma_2) = \theta\, Cstate' \wedge \forall\, Astate' : \gamma_2 \bullet \exists\, Astate : \gamma_1 \bullet AOp$$

The following example shows that the assumption on the domain of r is, in general, necessary. Consider an abstract data type with state space $\{0, \ldots, 3\}$, one operation $aop = \{(1, 2)\}$, and initial states 0 and 1. The concrete state space has just two points $\{0, 1\}$ and we are given a relation r as our retrieve relation, where $r = \{(\{0, 1\}, 0), (\{2\}, 1)\}$.

Then $cop = \mathrm{ran}(\mathrm{dom}\, aop \lhd_P r) \lhd (((l \setminus (l \fatsemi aop)) \fatsemi r)/(\mathrm{dom}\, aop \lhd_P r)) = \varnothing$. However, $cop = r^{-1} \fatsemi (l \setminus (l \fatsemi aop)) \fatsemi r = \{(0, 1)\}$. Therefore $r^{-1} \fatsemi (l \setminus (l \fatsemi aop)) \fatsemi r$ is not the most general powersimulation with this retrieve relation.

6 Conclusions

In this paper we have considered both the calculation and verification of refinements in state-based systems and in particular the Z specification language. We have derived a single complete refinement method in Z by using possibility mappings.

Although the prospect of arbitrary refinements needing several simulations might perhaps appear rather esoteric at first, it is interesting to note that this has arisen in an industrial context.

Recently an industrial refinement was carried out by a leading formal methods consultancy, and the refinement that was to be verified needed both a downward and an upward simulation (for various reasons). The refinement was eventually verified by using two simulations, and it would have been interesting to see if the construction discussed in this paper would have aided the verification process.

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 2(82):253–284, 1991.

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

[3] J. Derrick and E.A. Boiten. Calculating upward and downward simulations of state-based specifications. *Information and Software Technology*, 1999. To appear.

[4] 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.

[5] Kai Engelhardt and W-P de Roever. *Model-Oriented Data Refinement.* CUP, 1998.

[6] P.H.B. Gardiner and C. Morgan. A single complete rule for data refinement. *Formal Aspects of Computing*, 5:367–382, 1993.

[7] R. Gerth. Foundations of compositional program refinement. In J. W. de Bakker, W.-P de Roever, and G. Rozenberg, editors, *Stepwise refinement of distributed systems*, LNCS 430, pages 777–807. Springer-Verlag, 1990.

[8] 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.

[9] 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.

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

[11] N.A. Lynch. Multivalued possibility mappings. In J. W. de Bakker, W.-P de Roever, and G. Rozenberg, editors, *Stepwise refinement of distributed systems*, LNCS 430, pages 519–543. Springer-Verlag, 1990.

[12] Nancy Lynch and Frits Vaandrager. Forward and backward simulations for timing-based systems. In J. W. de Bakker, W. P. de Roever, C. Huizing, and G. Rozenberg, editors, *Real-Time: Theory in Practice (REX Workshop, Mook, The Netherlands, June 1991)*, LNCS 600, pages 397–446. Springer-Verlag, 1992.

[13] M. Merritt. Completeness theorems for automata. In J. W. de Bakker, W.-P de Roever, and G. Rozenberg, editors, *Stepwise refinement of distributed systems*, LNCS 430, pages 544–560. Springer-Verlag, 1990.

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

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

[16] 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.