Working Paper Series
ISSN 1177-777X

Guarded operations, refinement and simulation

Steve Reeves and David Streader

Working Paper: 02/2009
June 10, 2009

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Guarded operations, refinement and simulation

Steve Reeves and David Streader
Department of Computer Science, University of Waikato,
{dstr,stever}@cs.waikato.ac.nz

June 10, 2009

Abstract

Simulation rules have long been used as an effective computational means to decide refinement relations in state-based formalisms. Here we investigate how they might be amended so as to decide the event-based notion of singleton failures refinement of abstract data types or processes that have operations with a "guarded" interpretation.

As the results presented here and found elsewhere in the literature are so sensitive to the details of the definitions used, we have machine-checked our results.

1 Introduction

We will review some of the known results about abstract data type (ADT) refinement and simulation (and note that we have machine-checked those of interest to us). First, we need to note (much more is said on this later) that *state-lifted* data types are those where the local state of the data type has a special element added (usually denoted by $\perp$), and then the operations of the ADT are given meaning as relations over this *lifted* state. In contrast, *operation-lifted* ADTs are those where the state is lifted as previously, but each operation in the ADT is also lifted by adding $\perp$ to its domain and range and adding, according to various prescriptions to be illustrated later, new pairs to the relation that gives the meaning of the operation. In addition to the above liftings we can totalise too—this means that in either lifting case we require total relations as the outcome. These subtle differences between lifting operations lead to important results:

1. Hoare, He and Saunders (HHS)—forward and backward simulation sound and complete for state-lifted data type refinement [1]
2. Reeves and Streader—data refinement not equal to sF refinement [2]
3. Reeves and Streader—backward simulation is not sound w.r.t. sF refinement [3]
4. Derrick—one complete simulation rule for data refinement [4]
5. Derrick and Boiten—forward and backward simulation not complete with operation-lifted data types[5]

The single complete rule of [4], unlike the completeness of [1], does not require the construction of an intermediate ADT. In [4] the same power-set construction as used in the proof of completeness in [1] appears. But in [4] the "ADT" built by the power-set construction is simply a computational step in ascertaining if one ADT is indeed a refinement of another. Hence the result of the power-set construction need not satisfy the detailed definition of what constitutes an ADT.

As we know that data type refinement is not singleton failure refinement we will, subsequently, apply a similar analysis to singleton failure refinement with an amended definition of simulation and establish the following results:

1. amended forward and backward simulation are sound with respect to sF refinement
2. amended forward and backward simulation not complete for data types either with lifted state or lifted operations
3. amended forward and backward simulation are complete for data types with lifted state and singleton failure refinement
4. one complete simulation rule for singleton failure refinement

These results and those labelled with **Theorem** in the rest of the paper have been machine checked using Isabell [6].

The standard HHS result of soundness and joint completeness of forward and backward simulation with respect to refinement can be applied equally to partial relations and to total relations (and more specifically, to the total relations that are the outcome of lifting and totalising). The completeness proof involves the construction of an intermediate data type via a power-set construction.

The construction of the intermediate data type has been shown [5] to be very sensitive to the detailed definition of ADTs. Whether the result of the power-set construction is a valid data type or not depends on the definition of a data type you choose. With the completely reasonable **logical** definition chosen in [5] the output of the power-set construction is not a valid data type. With an alternative and more liberal definition (to be given later) the standard HHS result can be applied (and we have a (machine checked) completeness proof).

Consequently we have two possibilities: one, we can keep the logical definition [5] of data type with the consequence that the completeness proof fails; two, we can liberalise the definition of data types to include the results of the power-set construction and have a valid completeness proof.

2 Abstract data types with guarded operations

An ADT consists of a set of named operations that act on private state *State* plus two special operations:

init that initialises the data type by relating the public state to the private state and

final that terminates the data type by relating the private state back to the public state

All operations will be given a relational semantics.

Definition 1 *Simple Data Type* D*, where* $Names_{\mathsf{D}}$ *is a set of names for the operations of* D*,* $State_{\mathsf{D}}$ *is the local (private) state of* D *and* $State_{\mathsf{g}}$ *the global state of a program which uses* D*, is given by:*

$$(State_{\mathsf{D}}, Op_{\mathsf{D}}, \mathsf{init}_{\mathsf{D}}, \mathsf{final}_{\mathsf{D}})$$

and

$Op_{\mathsf{D}} : Names_{\mathsf{D}} \to State_{\mathsf{D}} \times State_{\mathsf{D}}$
$\mathsf{init}_{\mathsf{D}} : State_{\mathsf{g}} \times State_{\mathsf{D}}$
$\mathsf{final}_{\mathsf{D}} : State_{\mathsf{D}} \times State_{\mathsf{g}}$

We view this as saying that the operations in D *are named relations, so for the semantics of the (purely syntactic) operation name* $\mathsf{a} \in Names_{\mathsf{D}}$ *from ADT* D*, which we write* D.a *when we need to disambiguate, we write*

$$[\![\mathsf{D.a}]\!] \triangleq Op_{\mathsf{D}}(\mathsf{a})$$

For example, for ADT A the relations which give semantics to the operation names a, b and c are given by the solid lines in Fig. 1 (ignore anything involving $\perp$ for now).

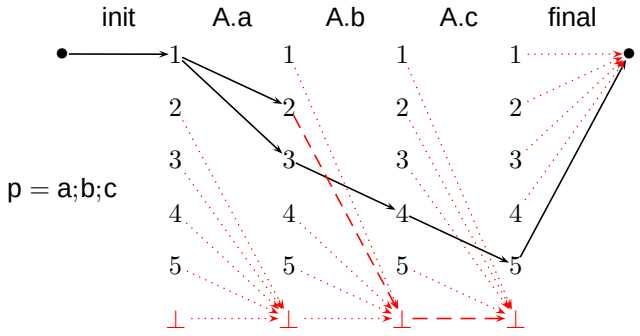

Figure 1: $[\![\mathsf{A.p}]\!]^{\mathsf{T}} \triangleq \mathsf{init}_{\mathsf{A}};[\![\mathsf{A.a}]\!]^{\mathsf{T}};[\![\mathsf{A.b}]\!]^{\mathsf{T}};[\![\mathsf{A.c}]\!]^{\perp};\mathsf{final}_{\mathsf{A}}$

The simple ADTs captured by Definition 1 are open to several informal interpretations. Their operations could be undefined outside of precondition (outside of the domain of the relation they denote) or they could be guarded outside of the precondition. In addition the behaviour inside the precondition could have a totally correct interpretation, i.e. the operation will terminate and will terminate in one of the post-states indicated by the relation or it could have a partially correct interpretation, i.e. the operation might terminate and if it terminates it will terminate in one of the post-states indicated by the relation.

One way to formalise the desired interpretation is to lift and totalise the (in general, partial) relation that gives the meaning of an operation name appropriately. A second way is to keep the original relations as the operations and define refinement that is consistent with (captures) the desired interpretation.

2.1 Lifting and totalising operations

We can lift and totalise the relations in many ways. Here we are interested in interpreting the operations:

1. as guarded outside of precondition
2. with a choice of termination interpretation:
   (a) the **total correctness interpretation**, i.e. they must terminate
   (b) the **partial correctness interpretation**, i.e. they may terminate

Point two is often not mentioned but because we are going to use $\bot$ to represent "not terminated" our semantics can explicitly distinguish operations that may terminate from operations that must terminate.

For example, in Fig. 1, if we consider the relations given by *all* the lines in the diagram, then we have lifted and totalised our operations to give them a guarded, total correctness meaning.

In contrast, the partially correct interpretation of guarded operations can be formalised by allowing an operation to always be able to terminate. Thus the relations relate all pre-states to $\bot$, indicating that termination is never guaranteed and hence it is always possible to not terminate. For example add $(1, \bot)$ to operation a in Fig. 1.

It is the exclusion of states from which an operation both might terminate and might fail to terminate that characterises the total correctness interpretation and which makes the the completeness proof of [1] fail.

We will formally define these possibilities for ADTs in the next section, but for now we introduce transformations on the semantics of single operations, like a, from some simple ADT D which reflect the above discussion.

First, the semantics that reflects guarded operations that must terminate (T for "total"):

$[\![\mathsf{D.a}]\!]^{\mathsf{T}} \triangleq [\![\mathsf{D.a}]\!] \cup \{(x, \bot) \mid x \in \mathit{State}_{\mathsf{D}} \cup \bot \wedge \neg \exists y.(x, y) \in [\![\mathsf{D.a}]\!]\}$

Secondly, the semantics that reflects guarded operations that may terminate (P for "partial"):

$[\![\mathsf{D.a}]\!]^{\mathsf{P}} \triangleq [\![\mathsf{D.a}]\!] \cup \{(x, \bot) \mid x \in \mathit{State}_{\mathsf{D}} \cup \bot\}$

2.2 Lifting and totalising data types

As we have indicated above, we have two ways to define extensions (completions?) of data types whose operations have a lifted and totalised relational semantics. We now give formal definitions for these alternatives.

Firstly we deal with *data types over lifted state*. That an ADT has been extended in this way is indicated by placing $\mathsf{S}_\bot$ as a superscript to the ADT name: this indicates that the relational semantics of the operations of the original simple ADT have been extended to give a new ADT over a lifted state. In order that we can lift this new value to whole programs (later) the global space is similarly lifted.

Definition 2 *Let* D *be some simple ADT* $(State_{\mathsf{D}}, Op_{\mathsf{D}}, \mathsf{init}_{\mathsf{D}}, \mathsf{final}_{\mathsf{D}})$. $\mathsf{D}^{\mathsf{S}_\perp}$ *is a state-lifted extension of* D. *The state-space* $State_{\mathsf{D}^{\mathsf{s}_\perp}} \triangleq State_{\mathsf{D}} \cup \{\perp\}$ *of* $\mathsf{D}^{\mathsf{S}_\perp}$ *contains a special value denoted by* $\perp$. $\mathsf{D}^{\mathsf{S}_\perp}$ *has the form*

$$(State_{\mathsf{D}^{\mathsf{s}_\perp}}, Op_{\mathsf{D}^{\mathsf{s}_\perp}}, \mathsf{init}_{\mathsf{D}^{\mathsf{s}_\perp}}, \mathsf{final}_{\mathsf{D}^{\mathsf{s}_\perp}})$$

where
$Op_{\mathsf{D}^{\mathsf{s}_\perp}} : Names_{\mathsf{D}} \to State_{\mathsf{D}^{\mathsf{s}_\perp}} \times State_{\mathsf{D}^{\mathsf{s}_\perp}}$
$\mathsf{init}_{\mathsf{D}^{\mathsf{s}_\perp}} : (State_{\mathsf{g}} \cup \perp) \times State_{\mathsf{D}^{\mathsf{s}_\perp}}$
$\mathsf{final}_{\mathsf{D}^{\mathsf{s}_\perp}} : State_{\mathsf{D}^{\mathsf{s}_\perp}} \times (State_{\mathsf{g}} \cup \perp)$
and
$Op_{\mathsf{D}^{\mathsf{s}_\perp}} \supseteq Op_{\mathsf{D}}$
$\mathsf{init}_{\mathsf{D}^{\mathsf{s}_\perp}} \supseteq \mathsf{init}_{\mathsf{D}}$
$\mathsf{final}_{\mathsf{D}^{\mathsf{s}_\perp}} \supseteq \mathsf{final}_{\mathsf{D}}$

Note that for state-lifted ADTs the operations (following the definition for simple ADTs) are, for any operation name $\mathsf{a} \in Names_{\mathsf{D}}$,

$$[\![\mathsf{D}^{\mathsf{S}_\perp}.\mathsf{a}]\!]^{\mathsf{S}} \triangleq Op_{\mathsf{D}^{\mathsf{s}_\perp}}(\mathsf{a})$$

Further note that there are no restrictions as to which relations are allowed as operations, save that they be total, including initialisation and finalisation.

Next, we deal with *data types with (explicitly) lifted operations*. Here the only relational semantics we admit as valid are those that are the result of a particular lifting of the operations of a simple abstract data type which is an example of either of the formalisations of must or may terminate from Section 2.1.

Definition 3 *Let* D *be some simple ADT* $(State_{\mathsf{D}}, Op_{\mathsf{D}}, \mathsf{init}_{\mathsf{D}}, \mathsf{final}_{\mathsf{D}})$. $\mathsf{D}^{\mathsf{OT}_\perp}$ *is an operation-lifted abstract data type with total correctness extension of* D. *The state-space* $State_{\mathsf{D}^{\mathsf{ot}_\perp}} \triangleq State_{\mathsf{D}} \cup \{\perp\}$ *contains a special value denoted by* $\perp$. $\mathsf{D}^{\mathsf{OT}_\perp}$ *has the form*

$$(State_{\mathsf{D}^{\mathsf{ot}_\perp}}, Op_{\mathsf{D}^{\mathsf{ot}_\perp}}, \mathsf{init}_{\mathsf{D}}^{\mathsf{OT}_\perp}, \mathsf{final}_{\mathsf{D}}^{\mathsf{OT}_\perp})$$

where
$Op_{\mathsf{D}^{\mathsf{ot}_\perp}} : Names_{\mathsf{D}} \to State_{\mathsf{D}^{\mathsf{ot}_\perp}} \times State_{\mathsf{D}^{\mathsf{ot}_\perp}}$
$\mathsf{init}_{\mathsf{D}}^{\mathsf{OT}_\perp} : (State_{\mathsf{g}} \cup \{\perp\}) \times State_{\mathsf{D}^{\mathsf{ot}_\perp}}$
$\mathsf{final}_{\mathsf{D}}^{\mathsf{OT}_\perp} : State_{\mathsf{D}^{\mathsf{ot}_\perp}} \times (State_{\mathsf{g}} \cup \{\perp\})$
and
$Op_{\mathsf{D}^{\mathsf{ot}_\perp}} \supseteq Op_{\mathsf{D}}$
$\mathsf{init}_{\mathsf{D}^{\mathsf{ot}_\perp}} = \mathsf{init}_{\mathsf{D}} \cup \{(\perp, \perp)\}$
$\mathsf{final}_{\mathsf{D}^{\mathsf{ot}_\perp}} = \mathsf{final}_{\mathsf{D}} \cup \{(\perp, \perp)\}$
and, for any operation a *from* $Names_{\mathsf{D}}$
$[\![\mathsf{D}^{\mathsf{OT}_\perp}.\mathsf{a}]\!] \triangleq [\![\mathsf{D}.\mathsf{a}]\!]^{\mathsf{T}}$

Definition 4 *Let* D *be some simple ADT* $(State_{\mathsf{D}}, Op_{\mathsf{D}}, \mathsf{init}_{\mathsf{D}}, \mathsf{final}_{\mathsf{D}})$. $\mathsf{D}^{\mathsf{OP}_\perp}$ *is an operation-lifted abstract data type with partial correctness extension of* D. *The state-space* $State_{\mathsf{D}^{\mathsf{op}_\perp}} \triangleq State_{\mathsf{D}} \cup \{\perp\}$ *contains a special value denoted by* $\perp$. $\mathsf{D}^{\mathsf{OP}_\perp}$ *has the form*

$$(State_{\mathsf{D}^{\mathsf{OP}_\perp}}, Op_{\mathsf{D}^{\mathsf{OP}_\perp}}, \mathsf{init}_{\mathsf{D}}^{\mathsf{OP}_\perp}, \mathsf{final}_{\mathsf{D}}^{\mathsf{OP}_\perp})$$

where

$Op_{\mathsf{D}^{\mathsf{OP}_\perp}} : Names_{\mathsf{D}} \rightarrow State_{\mathsf{D}^{\mathsf{OP}_\perp}} \times State_{\mathsf{D}^{\mathsf{OP}_\perp}}$
$\mathsf{init}_{\mathsf{D}}^{\mathsf{OP}_\perp} : (State_{\mathsf{g}} \cup \{\perp\}) \times State_{\mathsf{D}^{\mathsf{OP}_\perp}}$
$\mathsf{final}_{\mathsf{D}}^{\mathsf{OP}_\perp} : State_{\mathsf{D}^{\mathsf{OP}_\perp}} \times (State_{\mathsf{g}} \cup \{\perp\})$
and
$Op_{\mathsf{D}^{\mathsf{OP}_\perp}} \supseteq Op_{\mathsf{D}}$
$\mathsf{init}_{\mathsf{D}^{\mathsf{OP}_\perp}} = \mathsf{init}_{\mathsf{D}} \cup \{(\perp, \perp)\}$
$\mathsf{final}_{\mathsf{D}^{\mathsf{OP}_\perp}} = \mathsf{final}_{\mathsf{D}} \cup \{(\perp, \perp)\}$
and, for any operation a *from* $Names_{\mathsf{D}}$
$[\![\mathsf{D}^{\mathsf{OP}_\perp}.\mathsf{a}]\!] \triangleq [\![\mathsf{D}.\mathsf{a}]\!]^{\mathsf{P}}$

Clearly a data type with lifted operations is an example of a data type over lifted state. But there are data types over lifted state that are not a data type with lifted operations. Importantly the the data types built by the power-set construction, used in [1] to prove the completeness result are not ADT with lifted operations. The behaviour of lifted "must terminate" operations is restricted so that in any state they either can be performed and must terminate or cannot be performed and are blocked. Operations that from some state may be performed and terminate or may fail to terminate and are blocked do not satisfy the lifted operation definitions in Definition 3 and Definition 4.

3 Refinement and Simulation

A program p calls a sequence of operations each from some ADT[1]. This sequence must always start with init and end with final. For ease of writing init and final will often be omitted, but must be assumed to be present.

Definition 5 *If* $\{\mathsf{o}_\mathsf{i}\}_{1 \leq i \leq n}$ *are operation names from ADT* D *and* p *is the program* $\mathsf{o}_{\mathsf{i}_1}; \mathsf{o}_{\mathsf{i}_2}; \ldots; \mathsf{o}_{\mathsf{i}_m}$ *where* $1 \leq \mathsf{i}_\mathsf{j} \leq n$ *for* $1 \leq \mathsf{j} \leq m$ *then we say* p *is a program over* D *and*

$$\mathsf{D.p} \triangleq \mathsf{init}\ ;\ \mathsf{D.o}_{\mathsf{i}_1}\ ;\ \mathsf{D.o}_{\mathsf{i}_2}\ ;\ \ldots;\ \mathsf{D.o}_{\mathsf{i}_m}\ ;\ \mathsf{final}$$

We also extend the various ways of giving semantics to operation names to programs in the obvious way.

Definition 6 *If* $\{\mathsf{o}_\mathsf{i}\}_{1 \leq i \leq n}$ *are operation names from ADT* D *and* p *is the program* $\mathsf{o}_{\mathsf{i}_1}; \mathsf{o}_{\mathsf{i}_2}; \ldots; \mathsf{o}_{\mathsf{i}_m}$ *where* $1 \leq \mathsf{i}_\mathsf{j} \leq n$ *for* $1 \leq \mathsf{j} \leq m$ *then*

$$[\![\mathsf{D.p}]\!]^{\mathsf{X}} \triangleq \mathsf{init}_{\mathsf{D}}\ ;\ [\![\mathsf{D.o}_{\mathsf{i}_1}]\!]^{\mathsf{X}}\ ;\ [\![\mathsf{D.o}_{\mathsf{i}_2}]\!]^{\mathsf{X}}\ ;\ \ldots;\ [\![\mathsf{D.o}_{\mathsf{i}_m}]\!]^{\mathsf{X}}\ ;\ \mathsf{final}_{\mathsf{D}}$$

where X *can be any of* S, T *or* P *and the appropriate extensions of initialisation and finalisation for* X *are also used.*

[1] In what follows we allow "ADT" to range over all the possibilities for ADTs (simple or extensions) that we have seen so far.

Definition 7 *Data Refinement for guarded operations, written* $\sqsubseteq$ *(and possibly decorated with super- and sub-scripts), is dependent on the semantics (of the operations) of the two data types which it relates. If* A *and* C *are two data types and* p *is some program over those ADTs then*

$$\mathsf{A} \sqsubseteq^{\mathsf{X}} \mathsf{C} \triangleq [\![\mathsf{C.p}]\!]^{\mathsf{X}} \subseteq [\![\mathsf{A.p}]\!]^{\mathsf{X}}$$

where X *can be any of* S, T *or* P.

If we can construct a *simulation* on a partial relation semantics, either *forward* or *backward*, between the A and C above then we know there is a data refinement $\mathsf{A} \sqsubseteq \mathsf{C}$ from the well known soundness of simulation.

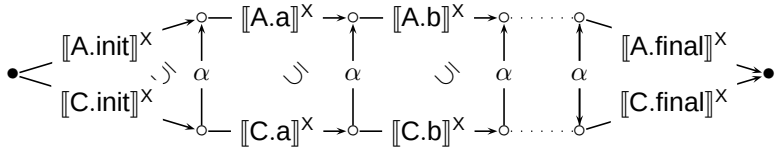

Figure 2: Backward simulation

Definition 8 *Let* A *and* C *be ADTs. There is a backward simulation relation between them iff there exists some* $\alpha \subseteq State_{\mathsf{C}} \times State_{\mathsf{A}}$ *such that*

1. $\mathsf{init_C};\alpha \subseteq \mathsf{init_A}$
2. $\forall\, \mathsf{o} \in Op_{\mathsf{A}}.[\![\mathsf{C.o}]\!]^{\mathsf{X}};\alpha \subseteq \alpha;[\![\mathsf{A.o}]\!]^{\mathsf{X}}$
3. $\mathsf{final_C};\alpha \subseteq \mathsf{final_A}$

Further, there is a forward simulation relation between A *and* C *iff there exists some* $\alpha \subseteq State_{\mathsf{C}} \times State_{\mathsf{A}}$ *such that*

F1. $\mathsf{init_C} \subseteq \alpha;\mathsf{init_A}$
F2. $\forall\, \mathsf{o} \in Op_{\mathsf{A}}.\alpha;[\![\mathsf{C.o}]\!]^{\mathsf{X}} \subseteq [\![\mathsf{A.o}]\!]^{\mathsf{X}};\alpha$
F3. $\mathsf{final_C} \subseteq \alpha;\mathsf{final_A}$

where X *can be any of* S, T *or* P *and the appropriate extensions of initialisation and finalisation for* X *are also used.*

Thus we have one definition for backward and one for forward simulation. These can be applied to both types of ADT the **state-lifted ADT** of Definition 2 and the **operation-lifted ADTs** of Definition 3 and Definition 4.

3.1 An aside

Before we move on, we make a remark that looks forward. The standard proof of soundness and completeness is based around the simple intuition of modelling operations as relations and the sequential composition of operations as relational composition (as we have done so far). But it is well known that modelling sequential composition of operations as the relational composition of possibly partial relations has a

meaning that *"differs from the meaning that would be be natural in a programming language"*, Spivey [7, p136]. This conceptual problem can be illustrated by looking back at Fig. 1 and considering the relational composition $[\![\mathsf{A.a}]\!]; [\![\mathsf{A.b}]\!]$ of the partial relations, the solid lines. $[\![\mathsf{A.a}]\!]; [\![\mathsf{A.b}]\!]$ is composed only of one element, the pair $(1, 4)$; the possibility of the $\mathsf{A.b}$ operation blocking after $\mathsf{A.a}$ performs the move $(1, 2)$ is lost in the construction of the relational composition. This problem clearly disappears if we restrict ourselves to total relations. Hence the usual way, which has been followed above, to avoid this well known pitfall is to lift and totalise the relational semantics prior to composition. But this introduces a new "state" $\perp$ to represent non-termination or not starting. We would note that because non-termination is unlike other states and cannot be observed directly it is often thought undesirable to include $\perp$ in the model.

3.2 Soundness and Completeness

We write $\sqsubseteq^{\mathsf{X}}_{\alpha}$ for backward simulation and $\sqsubseteq^{\mathsf{X}}_{\alpha^{-1}}$ for forward simulation. The Hoare, He and Saunders soundness result applies to all the various lifting and totalising regimes we have looked at above.

Theorem 1 *Soundness of forward and backward simulation [1]*

1. $\mathsf{A} \sqsubseteq^{\mathsf{X}}_{\alpha} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq^{\mathsf{X}} \mathsf{C}$

2. $\mathsf{A} \sqsubseteq^{\mathsf{X}}_{\alpha^{-1}} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq^{\mathsf{X}} \mathsf{C}$

Definition 9 *Forward and backward simulation are jointly complete iff there exist ADTs* $\mathsf{B}_1 \ldots \mathsf{B}_{n-1}$ *and there exist relations* $\alpha^1 \ldots \alpha^n$ *such that*

$$\mathsf{A} \sqsubseteq_{\alpha^1} \mathsf{B}_1 \sqsubseteq_{\alpha^2} \mathsf{B}_2 \ldots \sqsubseteq_{\alpha^n} \mathsf{C}$$

The important point is the existence of the intermediate data types and the relations. Hence this definition is dependent upon the chosen set of valid ADTs and the chosen set of valid relations.

The standard Hoare He and Saunders result [1] is that forward and backward simulation are sound and jointly complete certainly applies to the data types over lifted state. But as Boiten and Derrick [5] point out the joint completeness is not valid for what we call operation-lifted data types with the must terminate interpretation. It should be noted that the result fails because of the restriction placed on what operations are valid in the ADT and thus it is not always possible to compute chains of simulation between operation-lifted data types that refine each other. In order to regain the completeness property all we need to do is relax this restriction.

The power-set construction [8] builds an intermediate ADT (see Fig. 3). We have adopted the usual event-based convention and do not show the operations that are blocked; in state-based terminology these are operations that end at $\perp$.

Definition 10 *Power-set construction on operation-lifted semantics.*

Let A *be some operation-lifted must terminate ADT* $(State_{\mathsf{A}}, Op_{\mathsf{A}}, \mathsf{init}_{\mathsf{A}}, \mathsf{final}_{\mathsf{A}})$. *Let* $\bullet$ *be a member of the global state (we assume it is the only one since we need no more).*

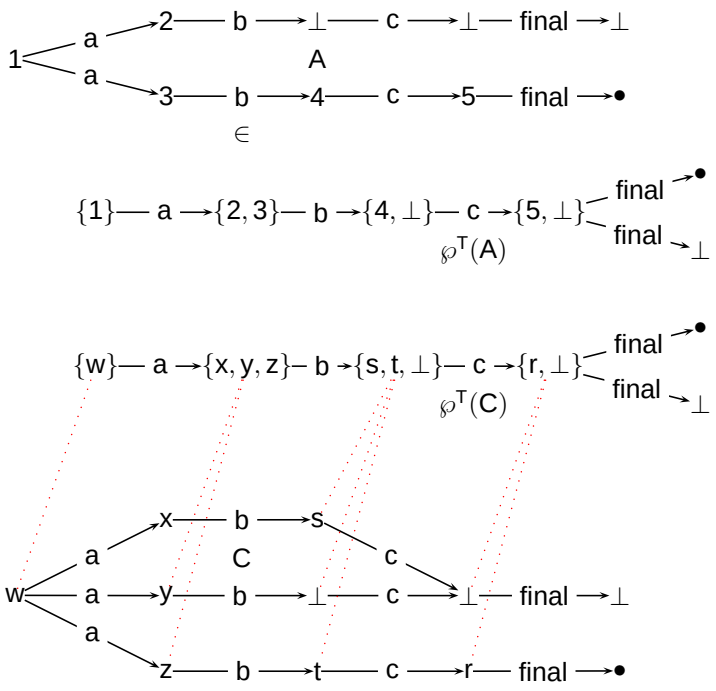

Figure 3: $\mathsf{A} \sqsubseteq \mathsf{C}$ as $\mathsf{A} \sqsubseteq^{\mathsf{T}}_{\in^{-1}} \wp^{\mathsf{T}}(\mathsf{A}) \sqsubseteq^{\mathsf{T}}_{\in} \wp^{\mathsf{T}}(\mathsf{C}) \sqsubseteq^{\mathsf{T}}_{\in} \mathsf{C}$

Let $R(X) \triangleq \{y \mid \exists x \in X.(x,y) \in R\}$. *In particular, for any operation name* $\mathsf{a} \in \mathit{Names}_{\mathsf{A}}$*:*

$$[\![\mathsf{a}]\!]^{\mathsf{T}}(X) \triangleq \{y \mid \exists x \in X.(x,y) \in [\![\mathsf{a}]\!]^{\mathsf{T}}\}.$$

Then,

$$\wp^{\mathsf{T}}(\mathsf{A}) \triangleq (\wp(\mathit{State}_{\mathsf{A}}), \wp^{\mathsf{T}}(\mathsf{init}_{\mathsf{A}}), \wp^{\mathsf{T}}(\mathit{Op}_{\mathsf{A}}), \wp^{\mathsf{T}}(\mathsf{final}_{\mathsf{A}}))$$

where

$\wp^{\mathsf{T}}(\mathsf{init}_{\mathsf{A}}) \triangleq (\bullet, \mathsf{init}_{\mathsf{A}}(\bullet))$
$\wp^{\mathsf{T}}(\mathit{Op}_{\mathsf{A}}) \triangleq \{(\mathsf{a}, (X, [\![\mathsf{a}]\!]^{\mathsf{T}}(X))) \mid X \subseteq \mathit{State}_{\mathsf{A}} \wedge \mathsf{a} \in \mathit{Names}_{\mathsf{A}}\}$
$\wp^{\mathsf{T}}(\mathsf{final}_{\mathsf{A}}) \triangleq \{(X, f) \mid X \subseteq \mathit{State}_{\mathsf{A}} \wedge f \in \mathsf{final}_{\mathsf{A}}(X)\}$

To show that forward and backward simulation are complete w.r.t. data refinement we apply the power-set construction to the (lifted, totalised) A and C thus building $\wp^{\mathsf{T}}(\mathsf{A})$ and $\wp^{\mathsf{T}}(\mathsf{C})$. A standard result is the existence of a backward simulation $\mathsf{A} \sqsubseteq^{\mathsf{T}}_{\in^{-1}} \wp^{\mathsf{T}}(\mathsf{A})$ and a forward simulation $\wp^{\mathsf{T}}(\mathsf{C}) \sqsubseteq^{\mathsf{T}}_{\in} \mathsf{C}$.

$$\mathsf{P} \sqsubseteq^{\mathsf{T}}_{\in^{-1}} \wp^{\mathsf{T}}(\mathsf{P}) \sqsubseteq^{\mathsf{T}}_{\in} \mathsf{P}$$

Thus P and $\wp^{\mathsf{T}}(\mathsf{P})$ are refine equivalent.

We can view the output from the power-set construction as a normal form and it can be seen that $\mathsf{A} \sqsubseteq \mathsf{C}$ if and only if $\wp^{\mathsf{T}}(\mathsf{A}) \sqsubseteq \wp^{\mathsf{T}}(\mathsf{C})$. Further we can rename the nodes used in the power-set construction so that $\wp^{\mathsf{T}}(\mathsf{A}) \sqsubseteq \wp^{\mathsf{T}}(\mathsf{C})$ if and only if when we ignore all unreachable states the concrete operations, including init and final, are a subset of the abstract operations with the same name $\forall o.\wp^{\mathsf{T}}(\mathsf{A.o}) \supseteq \wp^{\mathsf{T}}(\mathsf{C.o})$.

Thus both for data types with lifted operations which must terminate and data types over lifted state forward and backward simulation are sound and we have $\forall o.\wp^{\perp}(\mathsf{A.o}) \supseteq \wp^{\perp}(\mathsf{C.o})$ that act as a complete test for refinement.

3.3 The Logical Style of defining Simulation

There are two basic styles we can take when defining data simulation between ADT, the relational and the logical styles. The logical style usually makes use of pre- and post-condition predicates (hence our name for it). The pre-condition defines where the operation is defined (the image of its relational semantics) and the post-condition defines the relation between the initial and final state of an operation. For simulation on a simple ADT Definition 1 with no particular interpretation all that is needed in the logical style is the strengthening of the post-condition (remember we are dealing with guarded—blocking—semantics here).

Where the operations are to be interpreted as guarded outside of precondition and totally correct:

Relational style we lift (add $\perp$) and totalise the relational semantics and treat $\perp$ as part of the state space (Definition 8);

Logical style we define simulation as the preservation of the pre-condition and the strengthening of the post-condition.

This can be translated into conditions on the relational semantics and is often done in such a way that no reference to $\perp$ is needed. For people who are uneasy with the inclusion of $\perp$ ("what does $\perp$ really mean?") this is an advantage.

Definition 11 Logical style *Let* A *and* C *be ADTs. The is a backward simulation between them iff there exists some* $\alpha \subseteq State_{\mathsf{C}} \times State_{\mathsf{A}}$ *such that*

1. $\mathsf{init}_{\mathsf{C}};\alpha \subseteq \mathsf{init}_{\mathsf{A}}$
2. $\forall \mathsf{o} \in Op_{\mathsf{A}}.\overline{dom[\![\mathsf{C.o}]\!]} \subseteq \alpha^{-1}(\overline{dom[\![\mathsf{A.o}]\!]})$
3. $\forall \mathsf{o} \in Op_{\mathsf{A}}.[\![\mathsf{C.o}]\!];\alpha \subseteq \alpha;[\![\mathsf{A.o}]\!]$
4. $\mathsf{final}_{\mathsf{C}};\alpha \subseteq \mathsf{final}_{\mathsf{A}}$

Clause 2 is the preservation of the pre-condition or an *applicability* condition and clause 3 is the strengthening of the post-condition or a *correctness* condition.

With data types over lifted operations (Definition 3 and Definition 4) the logical and relational styles of simulation are the same. But when we use date types over lifted state (Definition 2) this is no longer true.

Remember there are data types over lifted state that are not data types with lifted operations. By looking at these extra data types we can see that Definition 8 is not the same as applying Definition 11 to a data type over lifted state.

Any logical style of simulation that characterised Definition 8 would, because of the extended set of relations allowable, require an additional explicit predicate to indicate that states were related to $\perp$.

From this logical perspective we can say that the cost of not allowing $\perp$ to be included in the predicates defining the behaviour of an operation is that the completeness result has been lost.

Theorem 2 *Completeness of simulation with respect to refinement for ADTs* A *and* C*:*

[1] *If* $\mathsf{A} \sqsubseteq \mathsf{C}$ *there exists a sequence of simulations between ADTs from* A *to* C

[5] $\mathsf{A} \sqsubseteq \mathsf{C} \Leftrightarrow \wp^{\mathsf{T}}(\mathsf{A}) \supseteq \wp^{\mathsf{T}}(\mathsf{C})$

The lack of completeness of forward and backward simulation for data types with lifted operations is important as it tells us that we cannot compute all refinements by constructing intermediate data types (with no explicit reference to non-termination) and computing forward or backward simulations. If on the other hand we permitted the definition of operations to make reference to non-termination then we would have the completeness result.

Alternatively tools like B and Event B could be amended to generate the proof obligations needed to establish if $\wp^{\mathsf{T}}(\mathsf{A}) \supseteq \wp^{\mathsf{T}}(\mathsf{C})$. This one rule is complete but such proof obligations may not be as easy to satisfy as the human-designed forward and backward simulations.

4 Is $\perp$ any more unobservable than any other state?

Elsewhere we have discussed, in much more detail, the distinct observations that are needed to define testing semantics that characterise data refinement as opposed to singleton failure refinement [9]. Here we do not need to be quite so abstract as we are only considering what can be observed in order to motivate the simulation relations we use.

Here we define contexts as being the set of states from which an operation can be called, often called the pre-condition, and observations as being what can validly be observed when an operation is executed from any given context.

If we assume that state and only state is observable then we might make the following set of four observations: initial state is *x*, initial state is *y*, final state is *x* and final state is *y*. From this we clearly have not effectively observed the behaviour of the operation. But if we observe traces of states then the following set of two observations, initial state is *x* followed by final state is *x*, and initial state is *y* followed by final state is *y*, then we have observed some of the behaviour of the operation.

We view the only possible observations of an operation to be complete traces, Tr^c, i.e. to be pre-state/post-state traces, or sub-traces thereof. Hence the range of the observation function (which given an operation tells us what we can observe of it) is

a subset of all possible traces of states that the operation can be in, which we write $range(Obs) \subseteq State^*$. Using these traces observations we are able view $\perp$, added by lifting, as no more than sugar for the actual observation of a short trace, i.e. a trace of length less than two.

The partial relation interpretation of some operation E can be modelled by restricting the observations to being a single pre-state/post-state pair (i.e. a sequence of length two) $Obs_{\mathsf{pr}} = Tr^c \mid_{State \times State}$ and using *State* as the set of contexts. To summarise:

Definition 12

$$\Xi_{\mathsf{pr}} \triangleq \{[\_]_x \mid x \in State\} \qquad Obs_{\mathsf{pr}} = Tr^c \mid_{State \times State}$$

For example, a partial relation interpretation of operation E over $State \triangleq \{x, y\}$ that maps the initial state *y*, to a final state *x* is represented by the partial relation:

$$[\![\mathsf{E}]\!]_{(\Xi_{\mathsf{pr}}, Tr^c \mid_{State \times State})} = \{(y, (y, x))\}.$$

Clearly the first observation of the initial state is redundant as it appears as the context and hence, without loss of detail or generality, we can drop this from the relational semantics. So our example can be written

$$[\![\mathsf{E}]\!]_{\mathsf{pr}} = \{(y, x)\}.$$

the usual partial relational semantics.

By expanding the contexts to include $()$ to represent *not started* and by not restricting the observations to being traces of length two, we are able to observe more behaviour:

Definition 13

$$\Xi_{\mathsf{Op}} \triangleq \{[\_]_x \mid x \in State \cup \{()\}\} \qquad Obs_{\mathsf{Op}} = Tr^c$$

•

Returning to the example operation E, which maps initial state $y \in State$ to a final state $x \in State$, let us assume we want to adapt it to model an operation that never terminates when started in *x* and always terminates when started in *y*. The empty trace of observations $()$ can be made only if the operation never starts, so we add $()$ to the set of contexts to represent not starting. Hence the relational semantics of our adapted example $[\![\mathsf{E}]\!]_{(\Xi_{\mathsf{Op}}, Tr^c)}$ is $\{(y, (y, x)), (x, (x)), ((), ())\}$ (see Fig. 4).

We transform our relational semantics into the more usual state-to-state semantics in two simple steps. Step 1: the trace of observations is a pre-state/post-state pair when E terminates and just the pre-state when E does not terminate. As the pre-state is known from the domain of the relation it, without loss of generality, can be removed (see $[\![\mathsf{E}]\!]_{Step1}$ in Fig. 4). Step two: by convention $()$ is represented by $\perp$ and hence the semantics of our example becomes $\{(y, x), (x, \perp), (\perp, \perp)\}$.

In Fig. 4 we have assumed:

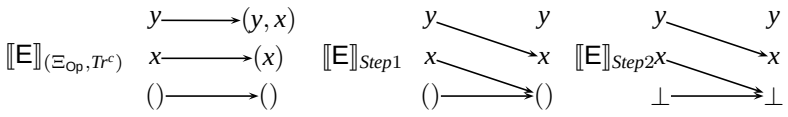

Figure 4: Semantics of operations

1. that nothing can be observed of an operation that has never been started hence $\bot$ maps to $\bot$ and only to $\bot$
2. the operation is *guarded outside of pre-condition* hence x is mapped to $\bot$ and only to $\bot$
3. we made a total correctness assumption, i.e. that E when started in state y will terminate in state x, hence $(y, \bot)$ is not in the semantics.

All-in-all this gives the semantics for an operation O to be:

$$[\![\mathsf{O}]\!] \triangleq [\![\mathsf{O}]\!]_{\mathsf{pr}} \cup \{(x, \bot)) \mid \neg\, \exists\, r.(x, r) \in [\![\mathsf{O}]\!]_{\mathsf{pr}}\} \cup \{(\bot, \bot)\}$$

which for our example is $\{(y, x), (x, \bot), (\bot, \bot)\}$

From the perspective outlined here an observation is not of state in isolation but traces of states and $\bot$ is just sugar for the observation of short traces (length less than two) and these short traces are just as observable as "normal" traces of length two. Because of this it can be argued that, from the perspective of what is observable, we have no pressing reason to exclude $\bot$ from a predicate used to define the observable behaviour of an operation.

With predicates that define both the terminating behaviour, i.e. observations of length two, and the non-terminating behaviour, i.e. observations of length less than two (predicates that include $\bot$), all we need in the logical style of simulation definition is the weakening of the post-condition.

4.1 Simulation relations and observations

The simulation relation α relates the abstract and concrete state. This state-to-state relation is then lifted to a relation that takes account of $\bot$. So, what restrictions, flowing from the discussion in the previous section, might there be on a simulation relation, if any?

Recall that we explain $\bot$ as a "short" trace, i.e. a trace which is either of length one or of length two: the former indicates a computation that has not started (and so it has not finished either), and the latter indicates a computation that has started but not finished. So, an operation will always map state $\bot$ to itself, because then we are asking what the operation does if it does not start, and it clearly does not finish, so the trace is $()$. What state should a simulation relate to $\bot$ when it indicates non-starting? It does not seem to be sensible for it to relate a non-starting state to anything but a non-starting state.

What of the situation where $\bot$ indicates an operation that did start, but which did not finish? Again, it does not seem sensible for a simulation to relate a situation in

which an operation does not finish when started in a certain state to anything but another operation starting in that state and also not finishing.

So, we would argue that a simulation can relate $\bot$, and only $\bot$, to only $\bot$.

If we accept the motivation we have just given and restrict the simulation relations as suggested then the completeness proof of forward and backward simulation fails even for state-lifted data types. This is because $\in$ and $\in^{-1}$ are no longer valid simulation relations. But from this alone we cannot conclude that there is no other sequence of restricted forward and backward simulation relating A to C or relating C to A in Fig. 3. It should be noted that the simulation relation is not simply a lifted relation, as can be seen by considering state $\{5, \bot\}$. In addition, the final operation is non-deterministic. The point being that if we restrict the simulation relation and/or the relational semantics of the operations (the final operation in this case) to being lifted relations, $[\![\_]\!]^{\bot}$, then the standard completeness proof fails.

Our restricted simulation relations become useful to us in the next section where we discuss singleton failure semantics, and from the results of the next section we can see the impossibility of relating A to C (Fig. 3) by the restricted simulation relations.

5 Singleton Failure semantics

First, a little notation: for ADT A let $s \xrightarrow{\mathsf{a}} \bot$ be event-based notation short for $(s, \bot) \in [\![\mathsf{a}]\!]^{\mathsf{T}}$ and where ρ is a sequence of operations let $s_{\mathsf{A}} \xrightarrow{\rho} s$ be notation for $(\bullet, s_{\mathsf{A}}) \in \mathsf{init}_{\mathsf{A}} \wedge (s_{\mathsf{A}}, s) \in [\![\rho]\!]^{\mathsf{T}}$ (so s_{A} is a start-state and ρ is a program).

Definition 14 *Singleton failures semantics of ADT* A *is given by sF where:*

$$sF(\mathsf{A}) \triangleq \{\{(\rho, \mathsf{a})\} \mid s_{\mathsf{A}} \in State_{\mathsf{A}} \wedge s_{\mathsf{A}} \xrightarrow{\rho} s \wedge s \xrightarrow{\mathsf{a}} \bot\} \cup \{(\rho, \{\}) \mid s_{\mathsf{A}} \in State_{\mathsf{A}} \wedge \exists s.s_{\mathsf{A}} \xrightarrow{\rho} s\}$$

Also, for ADTs A *and* C*,*

$$\mathsf{A} \sqsubseteq_{sF} \mathsf{C} \triangleq sF(\mathsf{C}) \subseteq sF(\mathsf{A})$$

Previously, using the processes in A and C,we have shown that backward simulation is not sound with respect to singleton failure semantics [2]. Here we will show that using the restricted simulation relations we can establish that forward and backward simulation are sound with respect to singleton failure semantics.

5.1 Sound restricted simulation relations

The definition of singleton failure semantics depends on the existence of a state *n* such that $n \neq \bot$ (the second element of the union in Definition 14) but this property is not preserved by the unrestricted simulation relations in the previous section, whereas the restricted simulation relations do preserve the property $n \neq \bot$, i.e. if $(x, y) \in \alpha$ then $x \neq \bot \Leftrightarrow y \neq \bot$.

Soundness with respect to singleton failure semantics follows for this restricted set of simulation relations.

Theorem 3 *Soundness of restricted simulation relation α between ADTs* A *and* C*:*

1. $\mathsf{A} \sqsubseteq^{\mathsf{T}}_{\alpha} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq_{sF} \mathsf{C}$
2. $\mathsf{A} \sqsubseteq^{\mathsf{T}}_{\alpha^{-1}} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq_{sF} \mathsf{C}$

From the soundness we can see that any sequence of simulations relate pairs of processes that are singleton failures refinements of each other only. Given the transitivity of singleton failure refinement we can see that by restricting the simulation relation we will never be able to relate A to C by simulation in Fig. 3 as it is easy to verify that $\mathsf{A} \not\sqsubseteq_{sF} \mathsf{C}$. But $\mathsf{A} \sqsubseteq \mathsf{C}$ (data refinement) *does* hold, hence with the restricted simulation relation forward and backward simulation are not complete with respect to data refinement, even if we use operation-lifted data types.

5.2 Completeness

To show completeness we apply a "guarded" power-set construction. This is the application of the power-set construction to the original states (i.e. not including $\bot$) only.

Definition 15 *Guarded power-set construction of on ADT* $\mathsf{A} \triangleq (\mathit{State}_{\mathsf{A}}, \mathsf{init}_{\mathsf{A}}, \mathsf{Op}_{\mathsf{A}}, \mathsf{final}_{\mathsf{A}})$ *restricted to* $\mathit{State}_{\mathsf{A}}$ *without* $\bot$ *is*

$$\begin{array}{l}
\wp(\mathsf{A}) \triangleq (\wp(\mathit{State}_{\mathsf{A}} \setminus \{\bot\}), \wp(\mathsf{init}_{\mathsf{A}}), \wp(\mathsf{Op}_{\mathsf{A}}), \wp(\mathsf{final})) \\
\wp(\mathsf{init}_{\mathsf{A}}) \triangleq (\bullet, \mathsf{init}_{\mathsf{A}}(\bullet)) \\
\wp(\mathsf{Op}_{\mathsf{A}}) \triangleq \{\wp(\mathsf{a}) \mid \mathsf{a} \in \mathit{Names}_{\mathsf{A}}\} \\
\wp(\mathsf{a}) \triangleq \{(X, [\![\mathsf{a}]\!](X)) \mid X \subseteq \mathit{State}_{\mathsf{A}} \setminus \{\bot\}\} \cup \{(X, \bot) \mid \exists x \in X.(x, \bot) \in [\![\mathsf{a}]\!]^{\mathsf{T}}\} \\
\wp(\mathsf{final}_{\mathsf{A}}) \triangleq \{(X, f) \mid X \subseteq \mathit{State}_{\mathsf{A}} \setminus \{\bot\} \wedge f \in \mathsf{final}_{\mathsf{A}}(X)\} \cup \{(\bot, \bot)\}
\end{array}$$

A program using the constructed data types has a deterministic path between elements of $\wp(\mathit{State})$, but with potential non-determinism where one branch ends at $\bot$. Just like for data refinement this construction requires data types from Definition 2 not Definition 3 or Definition 4. Hence our completeness proof, like that in Section 3 for ADT, is only correct for state-lifted ADT and is incorrect for operation-lifted ADT.

See Fig. 5 for an example of the use of the definition. Note that having restricted the simulation relations we are able to construct simulation relations between a restricted set of ADTs only. Although $\wp^{\mathsf{T}}(\mathsf{A}) \supseteq \wp^{\mathsf{T}}(\mathsf{C})$ (Fig. 3) we find that $\wp(\mathsf{A}) \not\supseteq \wp(\mathsf{C})$ (Fig. 5). This is just what we want as $\mathsf{A} \not\sqsubseteq_{sF} \mathsf{C}$ but $\mathsf{C} \sqsubseteq_{sF} \mathsf{A}$

The $\in$ relation between A and $\wp(\mathsf{A})$ preserves the singleton failures. From the Definition 15 it is easy to see that $\wp(\mathsf{A}) \supseteq \wp(\mathsf{C})$ if and only if $\wp(\mathsf{A}) \sqsubseteq_{sF} \wp(\mathsf{C})$.

Theorem 4 *There are completeness results for singular failures semantics that are similar to known results for data refinement. Let* A *and* C *be ADTs where* $\mathsf{A} \sqsubseteq_{sF} \mathsf{C}$*. Then:*

Similar to [1] *If* $\mathsf{A} =_{sF} \wp(\mathsf{A})$ *there exists a sequence of forward and backward simulations from* A *to* C

Similar to [5] $\wp(\mathsf{A}) \supseteq \wp(\mathsf{C})$ *if and only if* $\wp(\mathsf{A}) \sqsubseteq_{sF} \wp(\mathsf{C})$

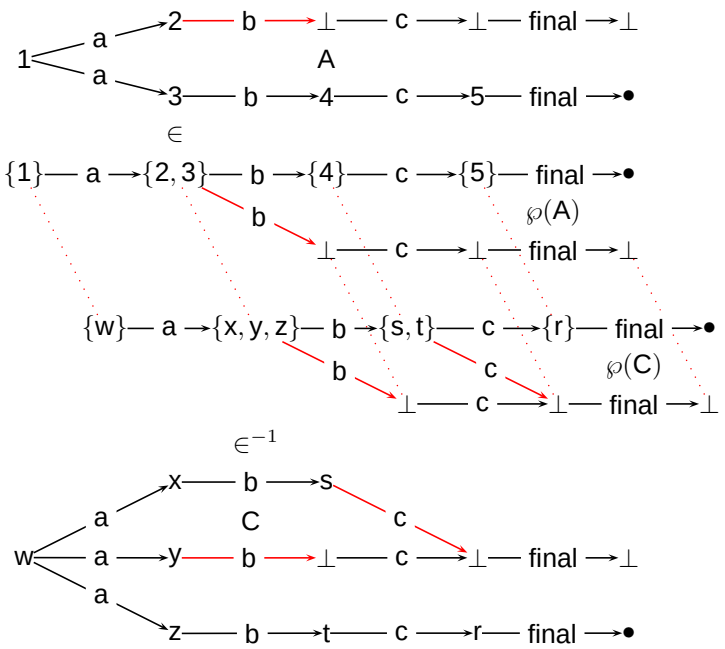

Figure 5: $A \sqsubseteq^{\in}_{L^{-1}} \wp(A)$ and $\wp(A) \not\supseteq \wp(C)$ but $\wp(C) \supseteq \wp(A)$

Theorem 4 provides us with a single complete rule for singleton failures semantics.

Althought our completeness proof is applicable to state-lifted ADT only, we are not asserting that an alternative approach might not provide a completeness proof for operation-lifted ADTs too. Constructing an intermediate ADT where all nondeterminism appears in the init operation appears a promising first step in the desigh of such a proof.

6 Conclusion

The known results for ADTs with guarded operations and a total correctness interpretation are:

State-lifted ADT have the properties:

1. forward and backward simulation are sound [1]
2. forward and backward simulation are jointly complete [1]
3. there is a single complete refinement rule: $A \sqsubseteq C \Leftrightarrow \wp^{\mathsf{T}}(A) \supseteq \wp^{\mathsf{T}}(C)$ [4]

Operation-lifted ADT have the properties:

1. forward and backward simulation are sound [1]

2. $\wp^{\mathsf{T}}(\mathsf{A})$ is not a data type with lifted operations and forward and backward simulation are **not** jointly complete [5]
3. there is a single complete refinement rule: $\mathsf{A} \sqsubseteq \mathsf{C} \Leftrightarrow \wp^{\mathsf{T}}(\mathsf{A}) \supseteq \wp^{\mathsf{T}}(\mathsf{C})$ [4]

The results for singleton failure semantics that we have machine checked are:

Singleton failures refinement for state-lifted ADT has the properties:

1. amended forward and backward simulation are sound
2. amended forward and backward simulation are jointly complete
3. there is an amended single complete refinement rule: $\mathsf{A} \sqsubseteq_{sF} \mathsf{C} \Leftrightarrow \wp(\mathsf{A}) \supseteq \wp(\mathsf{C})$

The amendment we have made to the definition of simulation relations has been motivated simply by considering what can be observed when an operation is executed (in Section 4).

References

[1] He, J., Hoare, C., Sanders, J.: Data refinement refined. ESOP 86 LNCS **213** (1986) 187–196

[2] Reeves, S., Streader, D.: Data refinement and singleton failures refinement are not equivalent. Formal Asp. Comput **20**(3) (2008) 295–301

[3] Reeves, S., Streader, D.: Must make seconf facs a tec report. Technical report, University of Waikato (2008) Computer Science Technical Report ??/2008, http://researchcommons.waikato.ac.nz/cms_papers/??/.

[4] Derrick: A single complete refinement rule for Z. JLC: Journal of Logic and Computation **10** (2000)

[5] Boiten, E., Derrick, J.: Incompleteness of relational simulations in the blocking paradigm. In ?? **??** (2008)

[6] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS. Springer (2002)

[7] Spivey, J.M.: The Z notation: A reference manual. 2nd. edn. Prentice Hall (1992)

[8] de Roever, W.P., Engelhardt, K.: Data Refinement: Model-Oriented Proof Methods and their Comparison. Cambridge Tracts in Theoretical Computer Science 47. Cambridge University Press (1998)

[9] Reeves, S., Streader, D.: General refinement, part one: interfaces, determinism and special refinement. In: Refine08 - International Refinement Workshop, Turku, Elsevier (2008) to appear.