Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 259 (2009) 177–191
www.elsevier.com/locate/entcs

Guarded Operations, Refinement and Simulation

Steve Reeves[1] David Streader[2]

Department of Computer Science
University of Waikato
Hamilton, New Zealand

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.

Keywords: refinement, simulation, guards

1 Introduction

In this paper we do two main things: we look at the consequences of two ways of lifting and completing relations with regard to refinement and simulation for abstract data types (ADTs) in a state-based world; we also look at the consequences of restricting what counts as a simulation in the context of the event-based view of refinement characterised by single failures refinement.

1.1 Data refinement and the state-based world

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 by

[1] Email:stever@cs.waikato.ac.nz
[2] Email:dstr@cs.waikato.ac.nz

doi:10.1016/j.entcs.2009.12.024

relations over this *lifted* state. In contrast, *operation-lifted* ADTs are those where each operation in the ADT is first given meaning by a relation over the (unlifted) state and then 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. Finally, the state is lifted as previously. 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 ways of lifting in ADTs lead to important results:

(i) forward and backward simulation are sound and jointly complete for state-lifted data type refinement [1]

(ii) a single complete simulation rule for data refinement [2]

(iii) forward and backward simulation are not jointly complete for data refinement with operation-lifted data types [3]

The standard result of soundness and joint completeness of forward and backward simulation with respect to refinement in [1] 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 single complete simulation rule of [2], unlike the joint completeness of [1], does not require the construction of an intermediate ADT. [2] uses the same power-set construction as in the proof of completeness in [1], but in [2] the structure built by the power-set construction is simply part of a computational step in ascertaining whether one ADT is indeed a refinement of another. Hence, in [2], the outcome of the power-set construction need not satisfy the definition of what constitutes an ADT.

The construction of the intermediate structure has been shown [3] to be very sensitive to the detailed definition of ADTs. Whether the outcome of the power-set construction is a valid data type or not actually depends on the definition of "data type" you choose. With the completely reasonable *logical* definition chosen in [3] the output of the power-set construction is not a valid data type (as we noted above). With an alternative and more liberal definition (to be given later) of "data type" the standard HHS result can be applied and we get a completeness proof again.

Consequently we have two possibilities: one, we can keep the logical definition [3] of data type with the consequence that the completeness proof fails; two, we can liberalise the definition of data type to include the results of the power-set construction and have a valid completeness proof.

It turns out that the two sorts of ADT differ in a very small way in their definitions and which is chosen is largely a matter of personal taste, and anyhow the literature already contains several subtly distinct definitions of ADTs.

1.2 Singleton failures refinement and the event-based world

We know that

(i) data refinement is not equal to singleton failures refinement [4]

(ii) backward simulation is not sound with respect to singleton failures refinement [5]

so we will analyse singleton failures refinement further (since it is not the same as data refinement it is interesting to see what its properties are). It will turn out that with a restricted definition of simulation we can establish the following results:

(i) restricted forward and backward simulation are sound with respect to singleton failures refinement and either sort of lifting

(ii) restricted forward and backward simulation are complete for singleton failures refinement and data types with lifted state

(iii) restricted forward and backward simulation are not complete for singleton failures refinement and data types with lifted operations

(iv) there is one complete simulation rule for singleton failures refinement and data types with either lifted state or lifted operations

(v) restricted forward and backward simulation are not complete for data refinement and data types with either lifted state or lifted operations

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

2 Abstract data types with guarded operations

An ADT consists of a set of named operations that act on private (local to the ADT) state *State* plus two special operations:

init that initialises the data type by relating the public (the global state in which the ADT is used) 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 2.1 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 (public) state of a program which uses D, is given by:

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

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 figure 1 (ignore anything involving $\bot$ for now).

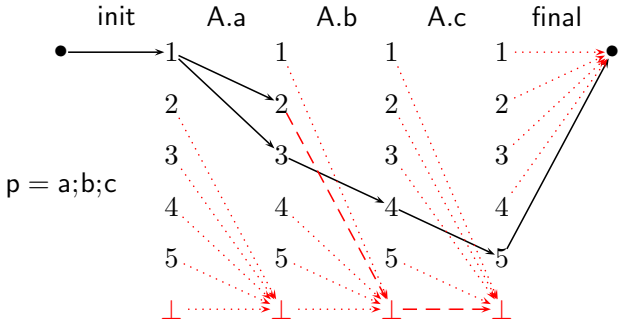

Fig. 1. $[\![\mathsf{A.p}]\!]^\mathsf{T} \triangleq \mathsf{init}_\mathsf{A};[\![\mathsf{A.a}]\!]^\mathsf{T};[\![\mathsf{A.b}]\!]^\mathsf{T};[\![\mathsf{A.c}]\!]^\bot;\mathsf{final}_\mathsf{A}$

The simple ADTs captured by definition 2.1 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:

(i) as guarded outside of precondition

(ii) 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 figure 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 not 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 figure 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 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 2.2 Let D be some simple ADT $(\mathit{State}_{\mathsf{D}}, \mathit{Op}_{\mathsf{D}}, \mathsf{init}_{\mathsf{D}}, \mathsf{final}_{\mathsf{D}}, \mathit{Names}_{\mathsf{D}})$. $\mathsf{D}^{\mathsf{S}_\bot}$ is a state-lifted extension of D. The state-space $\mathit{State}_{\mathsf{D}^{\mathsf{S}_\bot}} \triangleq \mathit{State}_{\mathsf{D}} \cup \{\bot\}$ of $\mathsf{D}^{\mathsf{S}_\bot}$ contains a special value denoted by $\bot$. $\mathsf{D}^{\mathsf{S}_\bot}$ has the form

$(\mathit{State}_{\mathsf{D}^{\mathsf{S}_\bot}}, \mathit{Op}_{\mathsf{D}^{\mathsf{S}_\bot}}, \mathsf{init}_{\mathsf{D}^{\mathsf{S}_\bot}}, \mathsf{final}_{\mathsf{D}^{\mathsf{S}_\bot}}, \mathit{Names}_{\mathsf{D}})$

where

$\mathit{Op}_{\mathsf{D}^{\mathsf{S}_\bot}} : \mathit{Names}_{\mathsf{D}} \to \mathit{State}_{\mathsf{D}^{\mathsf{S}_\bot}} \times \mathit{State}_{\mathsf{D}^{\mathsf{S}_\bot}}$

$\mathsf{init}_{\mathsf{D}^{\mathsf{S}_\bot}} : (\mathit{State}_{\mathsf{g}} \cup \{\bot\}) \times \mathit{State}_{\mathsf{D}^{\mathsf{S}_\bot}}$

$\mathsf{final}_{\mathsf{D}^{\mathsf{S}_\bot}} : \mathit{State}_{\mathsf{D}^{\mathsf{S}_\bot}} \times (\mathit{State}_{\mathsf{g}} \cup \{\bot\})$

and

$\forall \mathsf{a} \in \mathit{Names}_{\mathsf{D}}.\mathit{Op}_{\mathsf{D}^{\mathsf{S}_\perp}}(\mathsf{a}) \supseteq \mathit{Op}_{\mathsf{D}}(\mathsf{a})$
$\mathsf{init}_{\mathsf{D}^{\mathsf{S}_\perp}} \supseteq \mathsf{init}_{\mathsf{D}}$
$\mathsf{final}_{\mathsf{D}^{\mathsf{S}_\perp}} \supseteq \mathsf{final}_{\mathsf{D}}$
and all the relations are total.

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

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

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 2.3 Let D be some simple ADT $(\mathit{State}_{\mathsf{D}}, \mathit{Op}_{\mathsf{D}}, \mathsf{init}_{\mathsf{D}}, \mathsf{final}_{\mathsf{D}}, \mathit{Names}_{\mathsf{D}})$. $\mathsf{D}^{\mathsf{OT}_\perp}$ is an operation-lifted abstract data type with total correctness extension of D. The state-space $\mathit{State}_{\mathsf{D}^{\mathsf{OT}_\perp}} \triangleq \mathit{State}_{\mathsf{D}} \cup \{\perp\}$ contains a special value denoted by $\perp$. $\mathsf{D}^{\mathsf{OT}_\perp}$ has the form
$(\mathit{State}_{\mathsf{D}^{\mathsf{OT}_\perp}}, \mathit{Op}_{\mathsf{D}^{\mathsf{OT}_\perp}}, \mathsf{init}_{\mathsf{D}}^{\mathsf{OT}_\perp}, \mathsf{final}_{\mathsf{D}}^{\mathsf{OT}_\perp}, \mathit{Names}_{\mathsf{D}})$
where
$\mathit{Op}_{\mathsf{D}^{\mathsf{OT}_\perp}} : \mathit{Names}_{\mathsf{D}} \to \mathit{State}_{\mathsf{D}^{\mathsf{OT}_\perp}} \times \mathit{State}_{\mathsf{D}^{\mathsf{OT}_\perp}}$
$\mathsf{init}_{\mathsf{D}}^{\mathsf{OT}_\perp} : (\mathit{State}_{\mathsf{g}} \cup \{\perp\}) \times \mathit{State}_{\mathsf{D}^{\mathsf{OT}_\perp}}$
$\mathsf{final}_{\mathsf{D}}^{\mathsf{OT}_\perp} : \mathit{State}_{\mathsf{D}^{\mathsf{OT}_\perp}} \times (\mathit{State}_{\mathsf{g}} \cup \{\perp\})$
and
$\forall \mathsf{a} \in \mathit{Names}_{\mathsf{D}}.\mathit{Op}_{\mathsf{D}^{\mathsf{OT}_\perp}}(\mathsf{a}) \supseteq \mathit{Op}_{\mathsf{D}}(\mathsf{a})$
$\mathsf{init}_{\mathsf{D}^{\mathsf{OT}_\perp}} = \mathsf{init}_{\mathsf{D}} \cup \{(\perp, \perp)\}$
$\mathsf{final}_{\mathsf{D}^{\mathsf{OT}_\perp}} = \mathsf{final}_{\mathsf{D}} \cup \{(\perp, \perp)\}$
and, for any operation a from $\mathit{Names}_{\mathsf{D}}$, $[\![\mathsf{D}^{\mathsf{OT}_\perp}.\mathsf{a}]\!] \triangleq [\![\mathsf{D}.\mathsf{a}]\!]^{\mathsf{T}}$

Definition 2.4 Let D be some simple ADT $(\mathit{State}_{\mathsf{D}}, \mathit{Op}_{\mathsf{D}}, \mathsf{init}_{\mathsf{D}}, \mathsf{final}_{\mathsf{D}}, \mathit{Names}_{\mathsf{D}})$. $\mathsf{D}^{\mathsf{OP}_\perp}$ is an operation-lifted abstract data type with partial correctness extension of D. The state-space $\mathit{State}_{\mathsf{D}^{\mathsf{OP}_\perp}} \triangleq \mathit{State}_{\mathsf{D}} \cup \{\perp\}$ contains a special value denoted by $\perp$. $\mathsf{D}^{\mathsf{OP}_\perp}$ has the form
$(\mathit{State}_{\mathsf{D}^{\mathsf{OP}_\perp}}, \mathit{Op}_{\mathsf{D}^{\mathsf{OP}_\perp}}, \mathsf{init}_{\mathsf{D}}^{\mathsf{OP}_\perp}, \mathsf{final}_{\mathsf{D}}^{\mathsf{OP}_\perp}, \mathit{Names}_{\mathsf{D}})$
where
$\mathit{Op}_{\mathsf{D}^{\mathsf{OP}_\perp}} : \mathit{Names}_{\mathsf{D}} \to \mathit{State}_{\mathsf{D}^{\mathsf{OP}_\perp}} \times \mathit{State}_{\mathsf{D}^{\mathsf{OP}_\perp}}$
$\mathsf{init}_{\mathsf{D}}^{\mathsf{OP}_\perp} : (\mathit{State}_{\mathsf{g}} \cup \{\perp\}) \times \mathit{State}_{\mathsf{D}^{\mathsf{OP}_\perp}}$
$\mathsf{final}_{\mathsf{D}}^{\mathsf{OP}_\perp} : \mathit{State}_{\mathsf{D}^{\mathsf{OP}_\perp}} \times (\mathit{State}_{\mathsf{g}} \cup \{\perp\})$
and
$\forall \mathsf{a} \in \mathit{Names}_{\mathsf{D}}.\mathit{Op}_{\mathsf{D}^{\mathsf{OP}_\perp}}(\mathsf{a}) \supseteq \mathit{Op}_{\mathsf{D}}(\mathsf{a})$

$\mathsf{init}_{\mathsf{D}^{\mathsf{OP}_\perp}} = \mathsf{init}_{\mathsf{D}} \cup \{(\perp, \perp)\}$
$\mathsf{final}_{\mathsf{D}^{\mathsf{OP}_\perp}} = \mathsf{final}_{\mathsf{D}} \cup \{(\perp, \perp)\}$
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 data types built by the power-set construction, used in [1] to prove the completeness result, are not ADTs 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 2.3 and definition 2.4. So, any constructions containing such operations cannot be ADTs with lifted operations, hence the failure of the completeness proof.

3 Data refinement and simulation

A program p calls a sequence of operations each from some ADT [3]. 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 3.1 If $\{\mathsf{o}_\mathsf{i}\}_{1 \le i \le n}$ are operation names from ADT D and p is the program $\mathsf{o}_{\mathsf{i}_1}$; $\mathsf{o}_{\mathsf{i}_2}$; ....; $\mathsf{o}_{\mathsf{i}_m}$ where $1 \le \mathsf{i}_\mathsf{j} \le n$ for $1 \le \mathsf{j} \le m$ then we say p is a program over D and

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

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

Definition 3.2 If $\{\mathsf{o}_\mathsf{i}\}_{1 \le i \le n}$ are operation names from ADT D and p is the program $\mathsf{o}_{\mathsf{i}_1}$; $\mathsf{o}_{\mathsf{i}_2}$; ....; $\mathsf{o}_{\mathsf{i}_m}$ where $1 \le \mathsf{i}_\mathsf{j} \le n$ for $1 \le \mathsf{j} \le m$ then

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

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

Definition 3.3 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.

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

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.

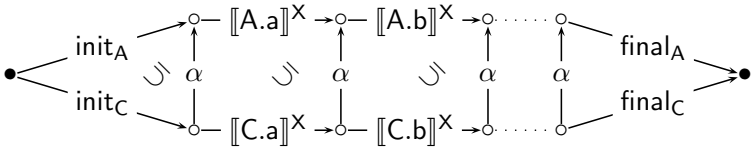

Fig. 2. Simulation

Definition 3.4 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

B1. $\mathsf{init}_{\mathsf{C}};\alpha \subseteq \mathsf{init}_{\mathsf{A}}$

B2. $\forall \mathsf{o} \in Op_{\mathsf{A}}.[\![\mathsf{C.o}]\!]^{\mathsf{X}};\alpha \subseteq \alpha;[\![\mathsf{A.o}]\!]^{\mathsf{X}}$

B3. $\mathsf{final}_{\mathsf{C}} \subseteq \alpha;\mathsf{final}_{\mathsf{A}}$

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. $\mathsf{init}_{\mathsf{C}} \subseteq \mathsf{init}_{\mathsf{A}};\alpha^{-1}$

F2. $\forall \mathsf{o} \in Op_{\mathsf{A}}.\alpha^{-1};[\![\mathsf{C.o}]\!]^{\mathsf{X}} \subseteq [\![\mathsf{A.o}]\!]^{\mathsf{X}};\alpha^{-1}$

F3. $\alpha^{-1};\mathsf{final}_{\mathsf{C}} \subseteq \mathsf{final}_{\mathsf{A}}$

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 ADTs of definition 2.2; and the operation-lifted ADTs of definition 2.3 and definition 2.4.

3.1 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 3.5 *Soundness of forward and backward simulation [1]*

(i) $\mathsf{A} \sqsubseteq^{\mathsf{X}}_{\alpha} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq^{\mathsf{X}} \mathsf{C}$

(ii) $\mathsf{A} \sqsubseteq^{\mathsf{X}}_{\alpha^{-1}} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq^{\mathsf{X}} \mathsf{C}$

Definition 3.6 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 what it means to be an ADT and what is an acceptable relation in this context.

The standard Hoare He and Saunders result [1] that forward and backward simulation are sound and jointly complete certainly applies to the data types over lifted state. But as Boiten and Derrick [3] 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.

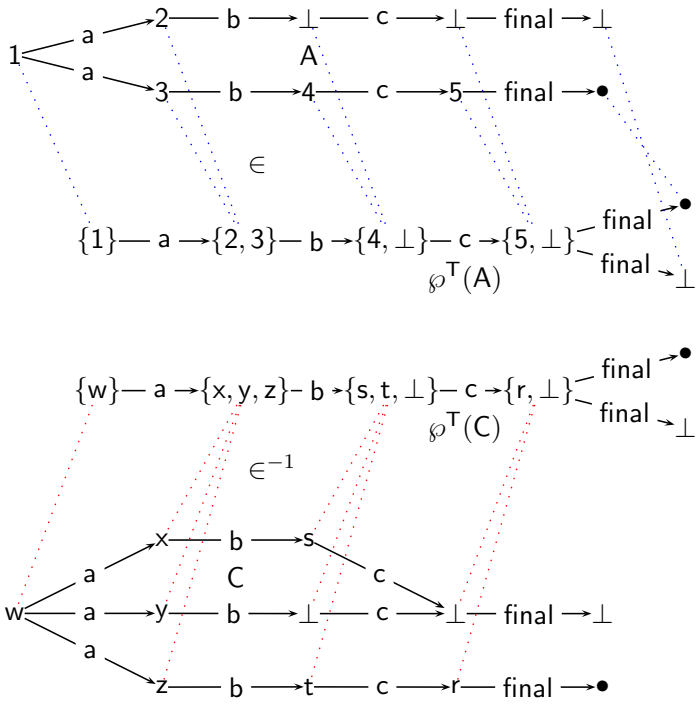

Fig. 3. $A \sqsubseteq C$ as $A \sqsubseteq^{\mathsf{T}}_{\in} \wp^{\mathsf{T}}(A) \sqsubseteq^{\mathsf{T}} \wp^{\mathsf{T}}(C) \sqsubseteq^{\mathsf{T}}_{\in^{-1}} C$

The power-set construction [8] builds an intermediate ADT (see figure 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 $\bot$.

Definition 3.7 Power-set construction on operation-lifted semantics.

Let A be some operation-lifted must-terminate ADT

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

Let $\bullet$ be a member of the global state (we assume it is the only one since we need

no more).

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

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

Then,

$$\wp^{\mathsf{T}}(\mathsf{A}) \triangleq (\wp(State_{\mathsf{A}}), \wp^{\mathsf{T}}(\mathsf{init}_{\mathsf{A}}), \wp^{\mathsf{T}}(Op_{\mathsf{A}}), \wp^{\mathsf{T}}(\mathsf{final}_{\mathsf{A}}), Names_{\mathsf{A}})$$

where

$$\wp^{\mathsf{T}}(\mathsf{init}_{\mathsf{A}}) \triangleq (\bullet, \mathsf{init}_{\mathsf{A}}(\bullet))$$
$$\wp^{\mathsf{T}}(Op_{\mathsf{A}}) \triangleq \{(\mathsf{a}, (X, [\![\mathsf{a}]\!]^{\mathsf{T}}(X))) \mid X \subseteq State_{\mathsf{A}} \wedge \mathsf{a} \in Names_{\mathsf{A}}\}$$
$$\wp^{\mathsf{T}}(\mathsf{final}_{\mathsf{A}}) \triangleq \{(X, f) \mid X \subseteq State_{\mathsf{A}} \wedge f \in \mathsf{final}_{\mathsf{A}}(X)\}$$

To show that forward and backward simulation are complete with respect to 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 (by construction) of a backward simulation $\mathsf{A} \sqsubseteq^{\mathsf{T}}_{\in} \wp^{\mathsf{T}}(\mathsf{A})$ and a forward simulation $\wp^{\mathsf{T}}(\mathsf{C}) \sqsubseteq^{\mathsf{T}}_{\in^{-1}} \mathsf{C}$.

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: $\forall\, o.\wp^{\mathsf{T}}(\mathsf{A.o}) \supseteq \wp^{\mathsf{T}}(\mathsf{C.o})$.

Theorem 3.8 *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

[3] $\mathsf{A} \sqsubseteq \mathsf{C}$ *iff* $\wp^{\mathsf{T}}(\mathsf{A}) \sqsubseteq \wp^{\mathsf{T}}(\mathsf{C})$

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

3.2 The Logical Style of defining simulation

There are two basic styles we can take when defining simulation between ADTs, the logical and the relational (as in the previous sections) 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 2.1) 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 $\bot$) and totalise the relational semantics and treat $\bot$ as part of the state space (definition 3.4);

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 $\bot$ is needed. For people who are uneasy with the inclusion of $\bot$ ("what does $\bot$ really mean?" is a common puzzle) this is an advantage.

Definition 3.9 Logical style Let A and C be ADTs. There is a backward simulation between them iff there exists some $\alpha \subseteq State_{\mathsf{C}} \times State_{\mathsf{A}}$ such that

(i) $\mathsf{init}_{\mathsf{C}};\alpha \subseteq \mathsf{init}_{\mathsf{A}}$

(ii) $\forall\, \mathsf{o} \in Op_{\mathsf{A}}.dom[\![\mathsf{C.o}]\!] \subseteq \alpha^{-1}(dom[\![\mathsf{A.o}]\!])$

(iii) $\forall\, \mathsf{o} \in Op_{\mathsf{A}}.[\![\mathsf{C.o}]\!];\alpha \subseteq \alpha;[\![\mathsf{A.o}]\!]$

(iv) $\mathsf{final}_{\mathsf{C}} \subseteq \alpha;\mathsf{final}_{\mathsf{A}}$

Clause (ii) is the preservation of the pre-condition or an *applicability* condition and clause (iii) is the strengthening of the post-condition or a *correctness* condition.

With data types over lifted operations (definition 2.3 and definition 2.4) the logical and relational styles of simulation are the same. But when we use data types over lifted state (definition 2.2) 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 using definition 3.4 is not the same as using definition 3.9 on a data type over lifted state.

Any logical style of simulation that characterised definition 3.4 would, because of the extended set of relations allowable, require an additional explicit predicate to indicate that some states were related to $\bot$.

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

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.

4 Singleton Failures 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 4.1 Singleton failures semantics of an 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 any ADTs A and C, $\mathsf{A} \sqsubseteq_{sF} \mathsf{C} \triangleq sF(\mathsf{C}) \subseteq sF(\mathsf{A})$

4.1 Sound restricted simulation relations

Previously, using the particular processes in A and C in figure 3, we have shown that backward simulation is not sound with respect to singleton failures semantics [4]. Here we will show that using restricted simulation relations we can establish that forward and backward simulation are sound with respect to singleton failures semantics.

The definition of singleton failures semantics depends on the existence of a state n such that $n \neq \bot$ (the second element of the union in definition 4.1) but this property is not preserved by the (usual) simulation relations as in the previous section, whereas *restricted simulation relations* do preserve the property.

Definition 4.2 A restricted simulation relation α is a simulation relation where if $(x, y) \in \alpha$ then $x \neq \bot \Leftrightarrow y \neq \bot$.

Soundness with respect to singleton failures semantics follows for this restricted notion of simulation relations.

Theorem 4.3 *Soundness. Let there be a restricted simulation relation α between ADTs A and C. Then:*

(i) $\mathsf{A} \sqsubseteq^{\mathsf{T}}_{\alpha} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq_{sF} \mathsf{C}$

(ii) $\mathsf{A} \sqsubseteq^{\mathsf{T}}_{\alpha^{-1}} \mathsf{C}$ *implies* $\mathsf{A} \sqsubseteq_{sF} \mathsf{C}$

From the soundness we can see that any sequence of simulations relates pairs of processes that are singleton failures refinements of each other only. Given the transitivity of singleton failures refinement we can see that we will never be able to relate A to C by a restricted simulation relation in figure 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.

4.2 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 4.4 Guarded power-set construction of on ADT $\mathsf{A} \triangleq (State_{\mathsf{A}}, \mathsf{init}_{\mathsf{A}}, \mathsf{Op}_{\mathsf{A}}, \mathsf{final}_{\mathsf{A}})$ where

$$\wp(\mathsf{A}) \triangleq (\wp(State_{\mathsf{A}} \setminus \{\bot\}) \cup \{\bot\}, \wp(\mathsf{init}_{\mathsf{A}}), \wp(\mathsf{Op}_{\mathsf{A}}), \wp(\mathsf{final}))$$
$$\wp(\mathsf{init}_{\mathsf{A}}) \triangleq (\bullet, \mathsf{init}_{\mathsf{A}}(\bullet))$$
$$\wp(\mathsf{Op}_{\mathsf{A}}) \triangleq \{\wp(\mathsf{a}) \mid \mathsf{a} \in Names_{\mathsf{A}}\}$$
$$\wp(\mathsf{a}) \triangleq \{(X, [\![\mathsf{a}]\!](X)) \mid X \subseteq State_{\mathsf{A}} \setminus \{\bot\}\} \cup \{(X, \bot) \mid \exists\, x \in X.(x, \bot) \in [\![\mathsf{a}]\!]^{\mathsf{T}}\} \cup \{(\bot, \bot)\}$$
$$\wp(\mathsf{final}_{\mathsf{A}}) \triangleq \{(X, f) \mid X \subseteq State_{\mathsf{A}} \setminus \{\bot\} \wedge f \in \mathsf{final}_{\mathsf{A}}(X)\} \cup \{(\bot, \bot)\}$$

A program using the constructed data types has a deterministic path between elements of $\wp(State)$, but with potential non-determinism where one branch ends at $\bot$. Just like for data refinement this construction requires data types from definition 2.2 not definition 2.3 or definition 2.4. Hence our completeness proof, like that in section 3 for ADT, is correct for state-lifted ADT only and is incorrect for operation-lifted ADT.

See figure 4 for an example of the use of the definition. Note that since the simulation relations are now restricted we are able to construct simulation relations between a restricted set of ADTs only. Although $\forall \mathsf{o}.\wp^{\mathsf{T}}(\mathsf{A.o}) \supseteq \wp^{\mathsf{T}}(\mathsf{C.o})$ (figure 3) we find that $\neg\ \forall \mathsf{o}.\wp(\mathsf{A.o}) \supseteq \wp(\mathsf{C.o})$ (figure 4). In particular, $\wp(\mathsf{A})$ does not contain and counterpart to $(\{s, t\}, \bot)$ in $\wp(\mathsf{C})$. This is just what we want as $\mathsf{A} \not\sqsubseteq_{sF} \mathsf{C}$ but $\mathsf{C} \sqsubseteq_{sF} \mathsf{A}$. (Note that $\in_\bot \triangleq \in \cup\, \{(\bot, \bot)\}$.)

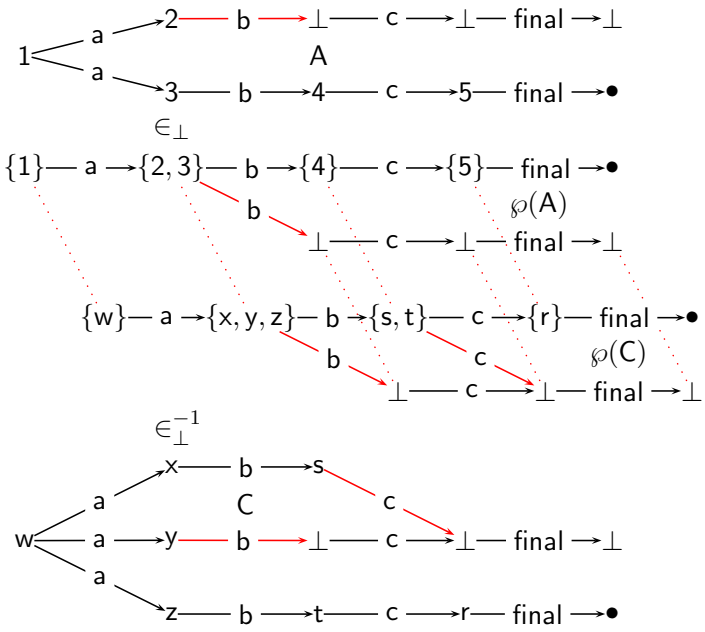

Fig. 4.

The $\in_\bot$ relation between A and $\wp(\mathsf{A})$ preserves the singleton failures, as does the $\in_\bot^{-1}$ relation between $\wp(\mathsf{C})$ and C.

From definition 4.4, *i.e.* by construction, it is easy to see that $\forall \mathsf{o}.\wp(\mathsf{A.o}) \supseteq \wp(\mathsf{C.o})$ if and only if $\wp(\mathsf{A}) \sqsubseteq_{sF} \wp(\mathsf{C})$.

Theorem 4.5 *Completeness of restricted simulation with respect to singleton failures refinement for ADTs* A *and* C*. (Similar to known results for data refinement.)*

Similar to [1] *If* $\mathsf{A} \sqsubseteq_{sF} \mathsf{C}$ *then there exists a sequence of restricted simulations between ADTs from* A *to* C

Similar to [3] $\mathsf{A} \sqsubseteq_{sF} \mathsf{C}$ *iff* $\wp(\mathsf{A}) \sqsubseteq_{sF} \wp(\mathsf{C})$

Theorem 4.5 provides us with a single complete rule for singleton failures semantics.

Although 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 design of such a proof.

5 Conclusion

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

State-lifted ADT have the properties:

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

Operation-lifted ADT have the properties:

(i) forward and backward simulation are sound [1]
(ii) $\wp^{\mathsf{T}}(\mathsf{A})$ is not a data type with lifted operations and forward and backward simulation are **not** jointly complete [3]
(iii) there is a single complete refinement rule: $\mathsf{A} \sqsubseteq \mathsf{C} \Leftrightarrow \wp^{\mathsf{T}}(\mathsf{A}) \sqsubseteq \wp^{\mathsf{T}}(\mathsf{C})$ [2]

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

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

(i) restricted forward and backward simulation are sound
(ii) restricted forward and backward simulation are jointly complete
(iii) there is a single complete refinement rule: $\mathsf{A} \sqsubseteq_{sF} \mathsf{C} \Leftrightarrow \wp(\mathsf{A}) \sqsubseteq_{sF} \wp(\mathsf{C})$

The restriction we have made to the definition of simulation relations has been motivated simply by considering what can be observed when an operation is executed.

References

[1] He, J., Hoare, C.A.R., Sanders, J.W.: Data refinement refined. In: Proc. of the European symposium on programming on ESOP 86, New York, NY, USA, Springer-Verlag New York, Inc. (1986) 187–196

[2] Derrick, J.: A single complete refinement rule for Z. Journal of Logic and Computation **10** (2000) 663–675

[3] Boiten, E., Derrick, J.: Incompleteness of relational simulations in the blocking paradigm. Draft (2008)

[4] Reeves, S., Streader, D.: General refinement, part two: flexible refinement. Proceedings of Refine 2008, Electronic Notes in Theoretical Computer Science (2008)

[5] Reeves, S., Streader, D.: State-based and event-based refinement: reconciling differences. Technical report, University of Waikato (2008) In http://researchcommons.waikato.ac.nz/cms.

[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 International series in computer science. Prentice Hall (1992)

[8] de Roever, W.P., Engelhardt, K.: Data Refinement: Model oriented proof methods and their comparison. Number 47 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (1998)

[9] Reeves, S., Streader, D.: General refinement, part one: interfaces, determinism and special refinement. Proceedings of Refine 2008, Electronic Notes in Theoretical Computer Science (2008)