Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 187 (2007) 35–53

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Relational Concurrent Refinement with Internal Operations

John Derrick[1]

Department of Computer Science, University of Sheffield, Sheffield, UK

Eerke Boiten[2]

Computing Laboratory, University of Kent, Canterbury, Kent, UK

Abstract

Data refinement in a state-based language such as Z is defined using a relational model in terms of the input-output behaviour of abstract programs. Downward and upward simulations form a sound and jointly complete methodology for verifying relational data refinements.
Refinement in a concurrent context, for example, as found in a process semantics, takes a number of different forms. Typically this is based on a notion of observation, for example, which events a system is prepared to accept or refuse. Concurrent refinement relations include trace refinement, failures-divergences refinement, readiness refinement and bisimulation.
In this paper we survey recent results linking the relational model of refinement to the process algebraic models. Specifically, we detail how variations in the relational framework lead to relational data refinement being in correspondence with traces-divergences, singleton failures and failures-divergences refinement in a process semantics. We then extend these results by showing how the effect of internal operations can be incorporated into the relational model. As a consequence simulation rules for failures-divergences refinement can be derived.

Keywords: Data refinement, Z, simulations, process algebraic semantics, failures-divergences refinement, internal operations.

1 Introduction

Motivated by both theoretical comparisons of refinement and integrations of specification languages, there has been recent interest in relating differing models of relational data refinement with process refinement relations arising in a concurrent

[1] Email: J.Derrick@dcs.shef.ac.uk

[2] Email: E.A.Boiten@kent.ac.uk

doi:10.1016/j.entcs.2006.08.043

context. The purpose of this paper is to survey and extend these results. In particular, we derive simulation rules for relational data refinement of specifications containing internal operations.

Models of relational refinement arise in contexts such as state-based specification languages like Z [19]. Specifications are considered to define abstract data types (ADTs), in which programs (sequences of operations) can be interpreted. Refinement in this context is taken to be the subset relation over program behaviours, where what is deemed visible is the input/output relation. Thus an ADT C refines an ADT A if for every program and sequence of inputs, the outputs that C produces are outputs that A could also have produced. *Simulations* have become the accepted approach to make verification of refinements tractable [10]. Theoretical background is given in [10], and examples of their use in Z are given in [21,11].

An alternative model is found in a process algebra where differing process semantics induce different refinement relations. For example, in CSP one could use trace refinement, failures refinement or failures-divergences refinement [17]. In CCS, bisimulation is typically used [16], whereas in LOTOS reduction, extension and conformance are defined [3]. A survey of many prominent refinement relations is given in [20]. These relations are often motivated by the description of an idealised machine by which one manipulates the system and observes its behaviour. Different concurrent refinement relations arise by varying the functionality of this machine.

In order to understand the nature and structure of refinement, as well as provide a means to combine specification languages and their development methodologies, it is necessary to understand the correspondence between data and process refinements. Work relating the two paradigms includes Josephs [15], He [14], Woodcock and Morgan [22], Bolton and Davies [5,6], Derrick and Boiten [2,12] and Schneider [18]. That due to Josephs [15], He [14], Woodcock and Morgan [22] defines a basic correspondence between simulation rules and failures-divergences refinement. The more recent work of Bolton and Davies [5,6], Derrick and Boiten [2,12] and Schneider [18] investigates a direct correspondence between the relational model and process semantics, and includes a specific consideration of input and output which introduces some subtleties.

The purpose of this paper is two-fold. First, we survey this existing work linking relational models of refinement to their process algebraic counterparts. Second, we extend these results (and, in particular, [12]) by showing how the effect of internal operations can be incorporated into the relational model.

The correspondence between a relational model and a process model can be investigated either by defining a 'corresponding process' in, say, CSP for each ADT, and then deriving the process semantics, or by defining a process semantics directly for an ADT. Schneider [18] and Bolton and Davies [5,6] do the former, whilst Derrick and Boiten the latter [12]. Either way, a process semantics $[\![\mathsf{A}]\!]$ can be given for an ADT A. The central aim is to derive results of the following form:

In relational model X, $\mathsf{A} \sqsubseteq_{data} \mathsf{C}$ if and only if $[\![\mathsf{A}]\!] \sqsubseteq_{ps} [\![\mathsf{C}]\!]$.

where $\sqsubseteq_{data}$ denotes some variation of relational data refinement, and $\sqsubseteq_{ps}$ the refinement relation induced by the given process semantics, the latter typically given as semantic models of CSP, for example, traces-divergences or failures-divergences.

The variations in the relational model include the interpretation of an operation given as a partial relation, and the observations made. Two possible interpretations are usually articulated for a partial operation: non-blocking and blocking. The former denotes a contract approach - outside a precondition anything may happen - the latter a behavioural approach - outside a precondition (guard) nothing may happen. The observations made in a relational model are usually restricted to the input/output of the ADT, however, these can be extended to include, for example, the refusals in a given state. We thus gain results such as:

> In the non-blocking relational model with standard observations, $\mathsf{A} \sqsubseteq_{\mathsf{data}} \mathsf{C}$ if and only if $[\![\mathsf{A}]\!]$ is traces-divergences refined by $[\![\mathsf{C}]\!]$.

This particular result is due to Schneider [18]. A version in a blocking model is due to Bolton and Davies, where the process semantics induced is a singleton-failures model. Derrick and Boiten [12] show what additional observations are needed to induce failures-divergences refinement, and derive corresponding simulation rules.

In this paper we extend these results with the consideration of internal (unobservable) operations, in particular deriving simulation rules for failures-divergences refinement. This is an important generalisation from previous approaches, particularly highlighting the role of divergence due to unbounded internal evolution and its effect on the simulation rules. We derive downward and upward simulation conditions for the blocking model, and as a consequence, relational refinement can be used as a basis for checking failures-divergences refinement of concurrent processes with internal operations. Thus, the verification of refinement using simulations can now be applied to all of CSP including hiding.

The paper is structured as follows. Section 2 highlights crucial elements of the background material, refering to other sources for the full details. Section 3 surveys work on relating refinement in relational and process semantics. Section 4 provides the extension of the blocking model to internal operations, and we conclude in Section 5.

2 Background

2.1 Relational refinement for Z

For the full details of abstract data type refinement in a relational setting and how this is used to develop a refinement theory for Z, both in the blocking and non-blocking approach, see [11] or the full version of this paper [13].

In summary, an abstract data type contains a (hidden) state space, initialisation, operations and a finalisation. A program corresponds to a sequence of operations, with initialisation and finalisation, representing a relation characterising observations on the visible state space. The definition of refinement between abstract data types requires (relational) inclusion of such observations for *all* programs, i.e., reduction of non-determinism. *Downward* and *upward* simulations form a sound and complete way of proving refinement by consideration of single operations.

The relational refinement theory comes in two equally valid flavours: for partial and for total relations. Here, we consider the total relation version only. Operations are often described by partial relations, which need to be embedded into total relations ("totalised") by the addition of fictitious elements to the observation space. This, also, comes in two variants, with corresponding interpretations. The *blocking* totalisation links states outside the operation's domain with a fictitious value *only*, indicating a state that the system should never reach – a property then to be preserved in refinement. The *non-blocking* totalisation links such states to *all* possible states including a fictitious one, indicating that any outcome is acceptable, thereby introducing non-determinism that may be reduced in refinement. In both approaches the simulation rules for partial operations are derived from those for totalised operations by removing references to the fictitious element.

Specifications in Z provide the additional complication that operations may have inputs and outputs. This is accommodated in the relational model by adding sequences of inputs and outputs to the visible and hidden state, with inputs provided at initialisation and all outputs observed at finalisation. Operations each consume one input and produce one output. The following definition, of the relational *finalisation* induced by a Z data type (which does not contain an explicit finalisation), is included fully, as it will be varied upon in the rest of this paper.

$$\mathsf{Fin} == \{State;\ is : \operatorname{seq} Input;\ os : \operatorname{seq} Output \bullet (is, os, \theta State) \mapsto (\langle\rangle, os)\}$$

The following (standard) definitions of refinement in Z follow from the constructions described above.

Definition 2.1 (Standard downward simulation in Z) Given Z data types $A = (AState, AInit, \{AOp_i\}_{i \in I})$ and $C = (CState, CInit, \{COp_i\}_{i \in I})$. The relation R on $AState \wedge CState$ is a *downward simulation* from A to C in the non-blocking model if

$$\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \wedge R'$$

and for all $i \in I$, $Input$ and $Output$:

$$\forall\, AState;\ CState \bullet \operatorname{pre} AOp_i \wedge R \Rightarrow \operatorname{pre} COp_i$$
$$\forall\, AState;\ CState;\ CState';\ \bullet \operatorname{pre} AOp_i \wedge R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i$$

In the blocking model, the final condition ("correctness") becomes

$$\forall\, AState;\ CState;\ CState';\ Input;\ Output \bullet R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i$$

Definition 2.2 (Standard upward simulation in Z)
Given Z data types $A = (AState, AInit, \{AOp_i\}_{i \in I})$ and $C = (CState, CInit, \{COp_i\}_{i \in I})$. Then the relation T on $AState \wedge CState$ is an *upward simulation* from A to C in the non-blocking model if

$$\forall\, CState \bullet \exists\, AState \bullet T$$
$$\forall\, AState';\ CState' \bullet CInit \wedge T' \Rightarrow AInit$$

and for all $i \in I$:

$$\forall\, CState \bullet \exists\, AState \bullet T \wedge (\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i)$$
$$\forall\, AState';\ CState;\ CState' \bullet$$
$$(COp_i \wedge T') \Rightarrow (\exists\, AState \bullet T \wedge (\mathrm{pre}\, AOp_i \Rightarrow AOp_i))$$

In the blocking model, the correctness condition becomes

$$\forall\, AState';\ CState;\ CState' \bullet (COp_i \wedge T') \Rightarrow \exists\, AState \bullet T \wedge AOp_i$$

2.2 Process refinement

A contrasting view of refinement is that offered by a process algebraic description of a system. There, instead of a relation over a global state being representative of a program, the traces of events (in essence, a record of all terminating programs) are recorded. There are a number of semantic models for CSP, each of which induces its own refinement relation.

Failures-divergences semantics The standard semantics of CSP is the failures-divergences semantics developed in [9,8,17]. A process is modelled by the triple $(A, \mathcal{F}, \mathcal{D})$ where A is its alphabet, $\mathcal{F}$ is its *failures* and $\mathcal{D}$ is its *divergences*. The failures of a process are pairs (t, X) where t is a finite sequence of events that the process may undergo and X is a set of events the process may refuse to perform after undergoing t. That is, if the process after undergoing t is in an environment which only allows it to undergo events in X, it may deadlock. The divergences of a process are the sequences of events after which the process may undergo an infinite sequence of internal events, i.e. livelock.

Failures and divergences are defined in terms of the events in the alphabet of the process. The failures of a process with alphabet A are a set

$$\mathcal{F} \subseteq A^* \times \mathbb{P}\, A$$

such that a number of properties hold: the sequences of events that a process can undergo form a non-empty, prefix-closed set; if a process can refuse all events in a set X then it can refuse all events in any subset of X; a process can refuse any event which cannot occur as the next event. The divergences of a process with alphabet A and failures $\mathcal{F}$ are a set $\mathcal{D} \subseteq \mathrm{dom}\, \mathcal{F}$ such that:

$$t_1 \in \mathcal{D} \wedge t_2 \in A^* \Rightarrow t_1 \frown t_2 \in \mathcal{D}$$
$$t \in \mathcal{D} \wedge X \subseteq A \Rightarrow (t, X) \in \mathcal{F}$$

These capture the idea that it is impossible to determine anything about a divergent process in a finite time. Therefore, the possibility that it might undergo further events cannot be ruled out. In other words, a divergent process behaves *chaotically*.

The failures-divergences semantics induces a refinement ordering defined in terms of failures and divergences [8]. A process Q is a refinement of a process P, denoted $P \sqsubseteq_{fd} Q$, if $\mathcal{F}(Q) \subseteq \mathcal{F}(P)$ and $\mathcal{D}(Q) \subseteq \mathcal{D}(P)$.

There are two other semantics models for CSP relevant to this paper.

Traces-divergences semantics The traces-divergences semantics is just the failures-divergences semantics with the refusal information removed. A process P is now modelled by $(A, \mathcal{T}, \mathcal{D})$, where $\mathcal{T}$ are the traces of P. It is obtained from the failures-divergences semantics by defining the traces as $\mathcal{T}(P) == \{tr \mid (tr, \varnothing) \in \mathcal{F}\}$.

The traces-divergences semantics induces a refinement ordering, where $P \sqsubseteq_{td} Q$ iff $\mathcal{T}(Q) \subseteq \mathcal{T}(P)$ and $\mathcal{D}(Q) \subseteq \mathcal{D}(P)$.

Singleton failures semantics The singleton failures semantics for CSP was used by Bolton [4] (and published in [5,6]) in order to define an appropriate correspondence with blocking data refinement. Essentially the singleton failures semantics is a failures semantics where the refusal sets have cardinality at most one. Specifically, a process is now modelled by $(A, \mathcal{S})$ where $\mathcal{S} \subseteq A^* \times \mathbb{P}_1 A$ (and $\mathbb{P}_1$ forms subsets of cardinality at most one).

If P is a process expressed in terms of *stop*, $\rightarrow$, $\sqcap$, $\Box$ and $\|$, then its singleton failures are given as the obvious projection from its failures, that is: $\mathcal{S}(P) = \mathcal{F}(P) \cap (A^* \times \mathbb{P}_1 A)$. This does not hold for processes containing hiding.

The singleton failures semantics induces a refinement ordering, where $P \sqsubseteq_{sf} Q$ iff $\mathcal{S}(Q) \subseteq \mathcal{S}(P)$.

Clearly, failures-divergences refinement is stronger than traces-divergences refinement, that is, $P \sqsubseteq_{fd} Q \Rightarrow P \sqsubseteq_{td} Q$. For divergent-free processes we have $P \sqsubseteq_{sf} Q \Rightarrow P \sqsubseteq_{td} Q$, and for divergent-free basic processes (i.e., ones expressed in terms of *stop*, $\rightarrow$, $\sqcap$, $\Box$ and $\|$) we have $P \sqsubseteq_{fd} Q \Rightarrow P \sqsubseteq_{sf} Q$. The relationship of singleton failures to other semantic models is discussed in [20] and later in [7].

3 Relating data and process refinement

The previous section outlined the standard relational theory of refinement as well as briefly mentioning relevant process based definitions. We now survey recent existing work relating relational refinement with process refinement. The correspondences are summarised in the following table.

Relational refinement	Process model	Citations
Non-blocking data refinement	traces-divergences	Schneider [18]
Blocking data refinement with deterministic outputs	singleton failures	Bolton and Davies [5,6]
Blocking data refinement	singleton failures of process and input process	Bolton and Davies [5,6]
Blocking data ref. with strengthened applicability but no input/output	failures	Josephs [15]
Blocking data ref. with extended finalisations	failures-divergences	Derrick and Boiten [2,12]
Non-blocking data ref. with extended finalisations but no input/output	failures-divergences	Derrick and Boiten [2,12]

Both Bolton and Davies and Schneider consider the standard definition of data refinement, which are thus verified by simulations as given (in Z) in Definitions 2.1 and 2.2. The 'extended finalisations' of Derrick and Boiten add conditions to the simulation rules, and these thus require augmented definitions.

3.1 Non-blocking data refinement and the traces-divergences semantics

Inspired by the work by Bolton and Davies [5,6] discussed below, Schneider [18] shows that non-blocking data refinement corresponds to traces-divergences refinement in a process semantics. To show this he translates ADTs into CSP directly, and uses the traces-divergences semantics on the resulting CSP process.

As with all approaches the result is first proved for ADTs without input and output, and then extended to the general case. The extension to ADTs with inputs and outputs (which Schneider calls communicating data types following Bolton) involves the embedding of input and output sequences in the global states similar to the one described above. For such an ADT the translation of an ADT A into a CSP process $process(A)$ is given by

$$process(A) == \sqcap s \in State, (*, s) \in Init \bullet Proc_A(s)$$

$$\begin{array}{l}
Proc_A(s) == \\
\quad \Box i \in I, in \in Input, (\langle in \rangle, \langle\,\rangle, s) \in \mathrm{dom}\, AOp_i \bullet \\
\quad\quad \sqcap s' \in State, out \in Output, (\langle in \rangle, \langle\,\rangle, s) \mapsto (\langle\,\rangle, \langle out \rangle, s') \in AOp_i \bullet \\
\quad\quad\quad\quad\quad\quad AOp_i.in.out \rightarrow Proc_A(s') \\
\quad \Box \\
\quad \Box i \in I, in \in Input, (\langle in \rangle, \langle\,\rangle, s) \notin \mathrm{dom}\, AOp_i \bullet \\
\quad\quad \sqcap out \in Output \bullet AOp_i.in.out \rightarrow \mathrm{div}
\end{array}$$

Note that here div is the divergent CSP process, which ensures that all events are possible after an operation has been called outside its precondition. The following (in the notation used in this paper) is then proved.

Theorem 3.1 *In the non-blocking model,* $A \sqsubseteq_{data} C$ *if and only if* $process(A) \sqsubseteq_{td} process(C)$.

3.2 Blocking data refinement and the singleton failures semantics

Bolton in [4] and Bolton and Davies in [5,6] discuss the relationship between data refinement and the singleton failures semantics [20] model. They consider both the blocking and non-blocking relational data type semantics, and, like Schneider, translate ADTs directly into CSP.

For the blocking model, the translation of an ADT A into a CSP process $\mathit{process}_b(A)$ is given by (using, for uniformity, the notation already introduced):

$$\mathit{process}_b(A) == \sqcap s \in \mathit{State}, (*, s) \in \mathit{Init} \bullet P_A(s)$$

$$\begin{array}{l} P_A(s) == \\ \quad \Box i \in I, \mathit{in} \in \mathit{Input}, (\langle \mathit{in} \rangle, \langle\,\rangle, s) \in \operatorname{dom} AOp_i \bullet \\ \qquad \sqcap s' \in \mathit{State}, \mathit{out} \in \mathit{Output}, (\langle \mathit{in} \rangle, \langle\,\rangle, s) \mapsto (\langle\,\rangle, \langle \mathit{out} \rangle, s') \in AOp_i \bullet \\ \qquad\qquad AOp_i.\mathit{in}.\mathit{out} \rightarrow P_A(s') \end{array}$$

As can be seen the enabling of this process is identical to that of Schneider, however, the effect of calling an operation outside its precondition is not now divergence but, since we are in the blocking model, simply inability to perform any event associated with that operation. This thus correctly reflects the intended meaning to the blocking model. For ADTs with deterministic outputs (or no input/output) the following is shown.

Theorem 3.2 *In the blocking model, for ADTs with deterministic outputs (or no input/output), $A \sqsubseteq_{\mathit{data}} C$ if and only if $\mathit{process}_b(A) \sqsubseteq_{\mathit{sf}} \mathit{process}_b(C)$.*

The inclusion of non-deterministic outputs complicates the process semantics needed, and an additional constraint is needed in order to characterise blocking data refinement. To do this a further partial translation is introduced, called *inputProcess* which provides a characterisation of when a particular input is in the domain. For the blocking model this is defined as:

$$\mathit{inputProcess}_b(A) == \sqcap s \in \mathit{State}, (*, s) \in \mathit{Init} \bullet P_A(s)$$

$$\begin{array}{l} P_A(s) == \Box i \in I, \mathit{in} \in \mathit{Input}, (\langle \mathit{in} \rangle, \langle\,\rangle, s) \in \operatorname{dom} AOp_i \bullet \\ \qquad\qquad \sqcap s' \in \mathit{State} \mid \\ \qquad\qquad\quad (\exists\, \mathit{out} \in \mathit{Output} \mid (\langle \mathit{in} \rangle, \langle\,\rangle, s) \mapsto (\langle\,\rangle, \langle \mathit{out} \rangle, s') \in AOp_i) \bullet \\ \qquad\qquad\qquad AOp_i.\mathit{in} \rightarrow P_A(s') \end{array}$$

As can be seen, this is the same as $\mathit{process}_b$ except that the outputs are unobservable. Blocking data refinement can then be shown to be equivalent to singleton failures refinement of both the process and the input process. That is:

Theorem 3.3 *In the blocking model, $A \sqsubseteq_{\mathit{data}} C$ if and only if $\mathit{process}_b(A) \sqsubseteq_{\mathit{sf}} \mathit{process}_b(C)$ and $\mathit{inputProcess}_b(A) \sqsubseteq_{\mathit{sf}} \mathit{inputProcess}_b(C)$.*

Two corollaries are worth noting. First, that this characterisation is equivalent to checking the singleton failures of the input process and trace refinement of the process. Second, that

$$\mathit{process}_b(A) \sqsubseteq_{fd} \mathit{process}_b(C) \Rightarrow A \sqsubseteq_{data} C$$

in the blocking model.

Bolton and Davies also consider the non-blocking model. However, the corresponding process used is different to that of Schneider. Specifically, they define $\mathit{process}_{nb}$ by

$$\mathit{process}_{nb}(A) == \sqcap s \in \mathit{State}, (*, s) \in \mathit{Init} \bullet Q_A(s)$$

$$\begin{array}{ll} Q_A(s) == & P_A(s) \\ & \Box \\ & ((\Box i \in I, \mathit{in} \in \mathit{Input}, (\langle \mathit{in} \rangle, \langle\,\rangle, s) \notin \operatorname{dom} AOp_i \bullet \\ & \qquad \sqcap s' \in \mathit{State}, \mathit{out} \in \mathit{Output} \bullet AOp_i.\mathit{in}.\mathit{out} \rightarrow \mathit{Chaos}) \\ & \quad \sqcap \mathit{stop}) \end{array}$$

where $\mathit{Chaos} == (\sqcap i \in I, \mathit{in} \in \mathit{Input}, \mathit{out} \in \mathit{Output} \bullet AOp_i.\mathit{in}.\mathit{out} \rightarrow \mathit{Chaos}) \sqcap \mathit{stop}$ is the non-divergent process that can perform any event, yet also refuse any event. In the process Q_A the lower *stop* is being used to represent non-termination of the operation. With this corresponding process analogous results are derived (i.e., data refinement corresponding to singleton as opposed to failures-divergences refinement). This non-blocking translation differs in key aspects from that defined by Schneider. In particular, the use of *Chaos* and *stop* to model non-termination seems a less natural embedding of non-blocking than using explicit divergence.

3.3 Defining a correspondence with failures-divergences refinement

Derrick and Boiten [2,12], motivated by the work of Bolton and Davies, explored what additional conditions were needed on relational refinement in order to achieve a correspondence with failures-divergences refinement. It turned out that explicit observation of refusals was the required ingredient, and since the finalisation determines what is observable, this involves generalising the finalisations used.

3.3.1 Basic construction

To add refusals to the observations of a data type, the finalisation used is generalised from being $\{\mathit{State} \bullet \theta\mathit{State} \mapsto *\}$ to becoming $\{\mathit{State} \bullet \theta\mathit{State} \mapsto E\}$. E will be a set of operation names representing the operations that could be refused at the state in which the finalisation is applied. Operations are indexed over I, and these are used as the set of operation names. Refusals E are any subset of the maximal refusals in a given state: $\{i : I \mid \neg \operatorname{pre} Op_i\}$. The full embedding is as follows[3].

Definition 3.4 (Refusals embedding) An ADT $(State, Init, \{Op_i\}_{i \in I})$ in the refusals interpretation is embedded in the relational model as follows. The global state G is $\mathbb{P}\, I$, finalisation is given by

$$\mathsf{Fin} == \{State;\ E : \mathbb{P}\, I \mid (\forall\, i \in E \bullet \neg \operatorname{pre} Op_i) \bullet \theta State \mapsto E\}$$

and initialisation is given by

$$\mathsf{Init} == \{Init;\ E : \mathbb{P}\, I \bullet E \mapsto \theta State'\}$$

The local state and the embedding of operations are unchanged.

Derrick and Boiten then derive two results. The first is to show what the consequence is on the simulation rules of this extended finalisation. The second is to show correspondence with failures-divergences refinement.

3.3.2 Simulation rules

The relational downward and upward simulation rules [11] contain conditions on the finalisations. Specifically, for a downward simulation that

$$\mathsf{R} \mathbin{⨾} \mathsf{CFin} \subseteq \mathsf{AFin}$$
$$\operatorname{ran}(\operatorname{dom} \mathsf{AFin} \lhd \mathsf{R}) \subseteq \operatorname{dom} \mathsf{CFin}$$

and for an upward simulation that

$$\mathsf{CFin} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AFin}$$
$$\forall\, \mathsf{c} : \mathsf{CState} \bullet \mathsf{T}(\!|\ \{\mathsf{c}\}\ |\!) \subseteq \operatorname{dom} \mathsf{AFin} \Rightarrow \mathsf{c} \in \operatorname{dom} \mathsf{CFin}$$

With the extended finalisation, where refusals are observed, the second downward simulation condition is always satisfied, and the first is equivalent to the standard downward applicability condition. Thus in the presence of refusals Definition 2.1 represents the correct formalisation.

For an upward simulation, $\forall\, \mathsf{c} : \mathsf{CState} \bullet \mathsf{T}(\!|\ \{\mathsf{c}\}\ |\!) \subseteq \operatorname{dom} \mathsf{AFin} \Rightarrow \mathsf{c} \in \operatorname{dom} \mathsf{CFin}$ is always satisfied, however, $\mathsf{CFin} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AFin}$ leads to a strengthening of the standard applicability condition from Definition 2.2 to

$$(1) \qquad \forall\, CState \bullet \exists\, AState \bullet \forall\, i : I \bullet T \land (\operatorname{pre} AOp_i \Rightarrow \operatorname{pre} COp_i)$$

As noted in [12]: 'The standard upward simulation applicability condition requires that we have to consider pairs of abstract and concrete states for each operation. The finalisation condition, on the other hand, requires that for every abstract state we can find a *single* concrete state such that all the preconditions of the abstract operations imply the preconditions of their concrete counterparts.'

[3] Since the result type of finalisation and the input of initialisation should both be G, the initialisation will take a set of operations as its input for consistency.

3.3.3 Correspondence with failures-divergences refinement

Bolton and Davies, and Schneider, both define the process semantics by first defining a 'corresponding process' in CSP and then using an appropriate semantics of this process. Derrick and Boiten, however, define a process semantics directly. The difference is not important, as long as the results are consistent.

With no outputs the traces, failures and divergences can be defined easily in either model, for the construction in the presence of input and output, see [1].

Blocking model Traces arise from sequences of operations which are defined within their guards. Refusals indicate the impossibility of applying an operation outside its precondition. Furthermore, there are no divergences since each operation is either blocked or gives a well-defined result.

Non-blocking model As no operation is blocked, every trace is possible: those that arise in the blocking model, and any others following divergence. There are no refusals beyond those after a divergence, as no operation is blocked, it either gives a well-defined result or causes divergence. There are now, however, divergences, which arise from applying an operation outside its precondition.

The following is then proved.

Theorem 3.5 *In both the non-blocking and the blocking model, relational refinement with extended finalisations corresponds to failures-divergences refinement.*

3.4 Discussion

We have seen that in both the non-blocking and blocking models it has been necessary to place additional restrictions (i.e., observations) on the standard definition of data refinement in order that failures-divergences refinement is achieved in a process semantics. Why this is is perhaps best illustrated via an example.

The non-blocking model. We have seen that without input/output non-blocking data refinement is equivalent to traces-divergences refinement. However, it is worth noting that this does not mean that data refinement suffers from the weakness of the CSP traces model. Specifically, although traces refinement is normally considered too weak since the deadlocked behaviour *stop* refines all processes, such a behaviour is not a feasible translation of an ADT. That is, no ADT will have corresponding process *stop*, since the non-blocking model allows all traces due to no operation being refused. In addition, unlike in trace refinement there is no bottom of the refinement ordering since all ADTs with all operations deterministic and fully defined have no strict refinements in this framework.

Without input/output, non-blocking data refinement is, in fact, also equivalent to failures-divergences refinement. To see this, note that without output the process semantics obtained identifies traces-divergences refinement and failures-divergences refinement, that is, $\mathit{process}(A) \sqsubseteq_{td} \mathit{process}(C)$ iff $\mathit{process}(A) \sqsubseteq_{fd} \mathit{process}(C)$. This is simply because there are no refusals (beyond those after a divergence) in the process semantics, since refusals only arise due to the presence of outputs.

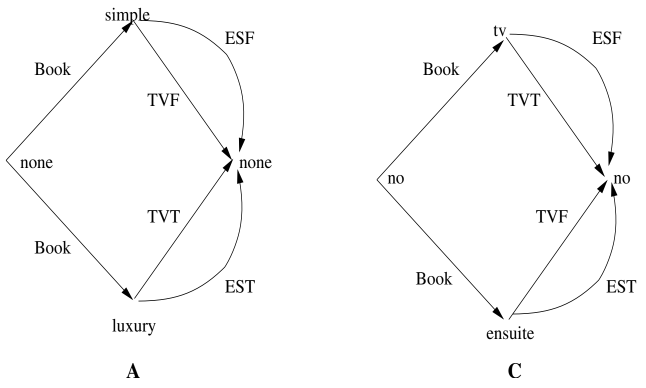

Fig. 1. Non-blocking, no input/output = traces-divergences and failures-divergences

Consider Figure 1, where in this and subsequent examples we define them via simple LTSs which represent the ADT's partial relations before totalisation.

These two specifications have the same traces and divergences, and are thus data refinement equivalent. There are no refusals, thus they are also failures-divergences equivalent. However, note that the stronger applicability condition needed for the blocking model does not hold here - for example for state *ensuite* it is not the case that any abstract state has

$$\forall\, i : I \bullet\ T \wedge (\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i)$$

The difference (i.e., why this does not matter in a traces-divergences model) is that with failures the refusals are tied to the traces, whereas for divergences we simply require their inclusion.

The blocking model. However, when considered under a blocking totalisation A and C are *not* failures-divergences equivalent. In fact, in a blocking scenario these are singleton failures equivalent and hence a blocking data refinement. To see this note that in a blocking model there are no divergences, the traces are the same in each. Now, although A has failure $(\langle Book\rangle, \{TVF, ESF\})$ which is not present in C, under a singleton failures model in A we just obtain singleton failures $(\langle Book\rangle, \{TVF\})$, $(\langle Book\rangle, \{ESF\})$, ... thus the difference is not observable. To recover failures-divergences refinement in the blocking model one needs to add the strengthened applicability condition. That is, it is precisely the condition

$$\forall\, CState \bullet \exists\, AState \bullet \forall\, i : I \bullet\ T \wedge (\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i)$$

that fails in this example.

4 Adding Internal Events to the Blocking model

In this section we consider refinement, and derive simulation rules, in the blocking model where divergences may arise from the existence of an internal, or unobservable, operation τ. As in the set-up described above, the observable operations, as well as τ, will be described as partial relations. We thus adapt the above totalisation, over which we can use the standard definition of data refinement.

We use the standard relational framework but now include an internal operation τ as an additional component of an ADT. From a (partial) abstract data type $\mathsf{S} == (\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i\in I}, \tau, \mathsf{Fin})$ where the finalisation Fin observes refusals as in Definition 3.4, our totalisation is

$$\widehat{\mathsf{S}}^{\mathsf{d}} == (\widehat{\mathsf{State}}^{\mathsf{d}}, \widehat{\mathsf{Init}}^{\mathsf{d}}, \{\widehat{\mathsf{Op}_i}^{\mathsf{d}}\}_{i\in I}, \widehat{\mathsf{Fin}}^{\mathsf{d}})$$

defined as follows.

State We take the standard blocking totalisation and add in another state, $\perp_{\mathsf{d}}$, to denote divergence.

$$\widehat{\mathsf{State}}^{\mathsf{d}} == \mathsf{State}_{\perp,\perp_{\mathsf{d}}} == \mathsf{State} \cup \{\perp, \perp_{\mathsf{d}}\}$$

Where $\perp, \perp_{\mathsf{d}} \not\in \mathsf{State}$, and $\perp \neq \perp_{\mathsf{d}}$. As these values will ultimately be observable, they will also be contained in the global state G.

Notation To define the totalised operations etc, we will use the following notations.

$\mathsf{State}\downarrow$ denotes stable states, i.e. those from which it is not possible to do an internal evolution. That is, $v \vdash \mathsf{State}\downarrow$ iff $v \vdash \neg\,\mathrm{dom}\,\tau$.

τ^* denotes finite internal evolution. It could be defined as the least fixed point of $\lambda\, R \bullet id \cup \tau \,⨾\, R$.

$\tau^{*|} == \tau^* \mathbin{⩥} (\mathrm{dom}\,\tau)$ is maximal finite internal evolution, leading to a stable state.

τ^ω denotes unbounded internal evolution, and, in particular, the set $\mathrm{dom}\,\tau^\omega$ consists of all the states from which unbounded internal evolution could start ("divergent states"). That set is defined as the largest fixed point of $\lambda\, S.\,\mathrm{dom}(\tau \rhd S)$. We also use the predicate $\mathsf{State}\uparrow$ to denote that a state is divergent, and $\mathsf{Init}\uparrow$ to indicate that the ADT has a divergent initial state.

$\widehat{\tau^\omega}$ encodes unbounded internal evolution as divergence, and is defined by

$$\widehat{\tau^\omega} == \mathrm{dom}\,\tau^\omega \times \mathsf{State}_{\perp,\perp_{\mathsf{d}}}$$

for the appropriate state space State.

$\mathsf{IE} == \tau^* \cup \widehat{\tau^\omega}$ encodes all internal evolution. In states where unbounded internal evolution is possible, the encoding through $\widehat{\tau^\omega}$ subsumes all finite internal evolution from such a state.

$\mathrm{div}\,\mathsf{Op}$ characterises all the states where the application of Op might be followed by unbounded internal evolution, and is defined by

$$\mathrm{div}\,\mathsf{Op} == \mathrm{dom}(\mathsf{Op} \,⨾\, \tau^\omega)$$

$\mathsf{DP} == \{\bot_{\mathsf{d}}\} \times \mathsf{State}_{\bot,\bot_{\mathsf{d}}}$ encodes preservation of divergence once it has occurred, using the appropriate state space State.

$\mathsf{BP} == \{(\bot,\bot)\}$ encodes the preservation of blocking.

Initialisation The totalisation of the initialisation will correspond to the initialisation followed by any potential internal evolution due to τ, encoded as divergence where unbounded. BP and DP are included to ensure totality of the initialisation.

$$\widehat{\mathsf{Init}}^{\mathsf{d}} == \mathsf{Init} \mathbin{⨾} \mathsf{IE} \cup \mathsf{BP} \cup \mathsf{DP}$$

Operations The totalisation of a partial relation Op is achieved in two stages. First, the blocking totalisation is applied, yielding an operation $\widehat{\mathsf{Op}}^{\mathsf{b}}$ total on $\mathsf{State}_{\bot}$. Second, we form $\widehat{\mathsf{Op}}^{\mathsf{d}} : \mathsf{State}_{\bot,\bot_{\mathsf{d}}} \leftrightarrow \mathsf{State}_{\bot,\bot_{\mathsf{d}}}$ which appends any potential evolution due to τ, encoded as divergence where it is unbounded.

$$\widehat{\mathsf{Op}}^{\mathsf{d}} == (\widehat{\mathsf{Op}}^{\mathsf{b}} \mathbin{⨾} \mathsf{IE}) \cup \mathsf{DP}$$

We thus absorb the effect of τ into the operations and initialisation, and represent its behaviour as non-determinism and divergence.

Finalisation We assume the partial ADT's finalisation already records refusals as in Definition 3.4. In order to achieve a correspondence with the *stable* failures model, we need to ensure that refusals are only recorded in stable states, i.e., ones where τ is not enabled. To this purpose, we prepend maximal internal behaviour to the finalisation. If the observed state was divergent already, this will have been taken into account by the previous operation (or initialisation if none). Otherwise, we record all refusals in all stable states reachable from the current one through internal evolution [4].

$$\widehat{\mathsf{Fin}}^{\mathsf{d}} == (\tau^{*|} \mathbin{⨾} \mathsf{Fin}) \cup \mathsf{BP} \cup \mathsf{DP}$$

Data refinement Data refinement is defined via the totalisation, that is, $\mathsf{A} \sqsubseteq_{data} \mathsf{C}$ iff $\widehat{\mathsf{A}}^{\mathsf{d}} \sqsubseteq_{data} \widehat{\mathsf{C}}^{\mathsf{d}}$.

4.1 The correspondence with failures-divergences refinement

We next need to define the failures-divergences semantics of an abstract data type, that is, its stable failures and divergences [5]. This can be defined either directly, or via a corresponding CSP process (e.g., as in [5,18]). We do the former. We denote the failures-divergences semantics $(\alpha(\mathsf{A}), \mathcal{F}(\mathsf{A}), \mathit{Div}(\mathsf{A}))$ of a data type A by $[\![\mathsf{A}]\!]$.

[4] Note that this is not strictly necessary as those states would already be part of the considerations for the semantics of the same trace; however, the alternative of making the finalisation a *partial* relation, though probably harmless in the end, takes us outside the constraints of the underlying theory [11].

[5] Note that the traces in this semantic model can be derived from the failures.

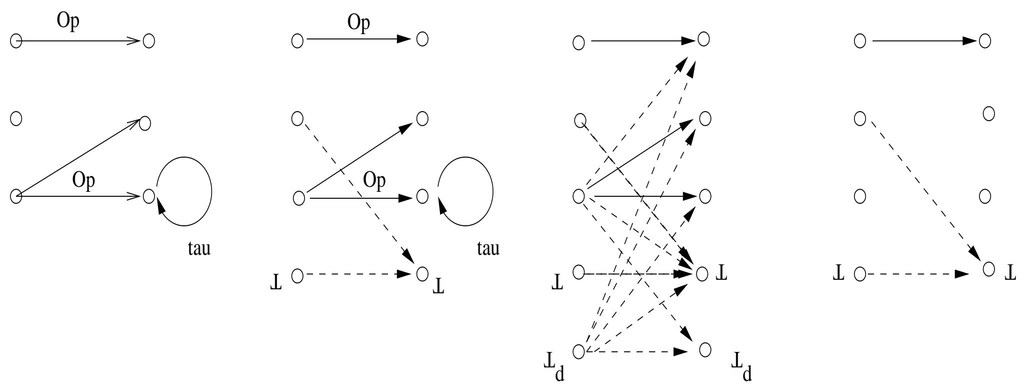

Fig. 2. Op, $\widehat{\mathsf{Op}}^{\mathsf{b}}$, $\widehat{\mathsf{Op}}^{\mathsf{d}}$ and $\widehat{\mathsf{Op}}^{\mathsf{b}} \setminus \mathsf{divOp}$

The alphabet $\alpha(\mathsf{A})$ is the set of operation indices I. Every trace is a sequence of indices. A (non-divergent) trace represents a computation (non-empty relation) consisting of an initialisation, followed by the corresponding sequence of operations, interleaved with finitely many internal operations. That is, $< i_1, \ldots, i_n >$ is a trace of A iff $\exists_{j=0\ldots n} \mathsf{State}_j \bullet \mathsf{State}_0 \in \mathrm{ran}(\mathsf{Init} \mathbin{⨾} \tau^*) \wedge \forall\, k : 1..n \bullet (\mathsf{State}_{k-1}, \mathsf{State}_k) \in \mathsf{Op}_{i_k} \mathbin{⨾} \tau^*$. If the data type is clear from the context, we identify the trace with this relation and write, for example, $(\mathsf{g}, \mathsf{State}') \in tr$ or $\mathsf{State}' \in \mathrm{ran}\, tr$.

Divergences are traces from which an infinite sequence of internal operations are feasible. A trace tr is divergent (denoted $tr \uparrow$) whenever there exist a prefix tr' of tr and a state $\mathsf{State}' \in \mathrm{ran}\, tr'$ such that $\mathsf{State}' \in \mathsf{State} \uparrow$.

Stable failures are pairs (tr, X), where tr is a trace, and a stable state $\mathsf{State}' \in \mathrm{ran}\, tr$ exists such that $\forall\, i \in X \bullet \mathsf{State}' \vdash \neg\, \mathrm{dom}\, \mathsf{Op}_i$. Every non-divergent trace leads to at least one such stable state.

Theorem 4.1 *In the blocking model,* $\mathsf{A} \sqsubseteq_{data} \mathsf{C}$ *iff* $[\![\mathsf{A}]\!] \sqsubseteq_{fd} [\![\mathsf{C}]\!]$.

Proof. See [13]. □

4.2 The simulation rules

We have shown that under the process semantics given to abstract data types, in a context where we have included internal events and their potential divergence, relational refinement corresponds to failures-divergences refinement. It remains now to extract simulation rules that can be applied to the schema calculus, since we do not wish to work with sets of events as currently embedded in the finalisation, nor explicit calculation of divergent traces.

We use the standard method to describe simulation rules on the underlying partial relations for verifying data refinement with extended finalisations. The key to doing this is to see that $\widehat{\mathsf{Op}}^{\mathsf{d}}$ is constructed as two totalisations. The first, $\widehat{\mathsf{Op}}^{\mathsf{b}}$, the standard blocking totalisation, the second, where we add in $\perp_{\mathsf{d}}$ is, in fact, the standard non-blocking totalisation on $\widehat{\mathsf{Op}}^{\mathsf{b}} \setminus \mathrm{div}\, \mathsf{Op}$. See Figure 2.

Let us denote $\widehat{\mathsf{Op}}^{\mathsf{b}} \setminus \operatorname{div} \mathsf{Op}$ by $\widehat{\widehat{\mathsf{Op}}}$.

4.2.1 Downward simulations

We consider simulation relations $\widetilde{\mathsf{R}}$ on the extended state spaces defined in terms of simulations R between the basic state spaces, as follows:

$$\widetilde{\mathsf{R}} == \mathsf{R} \cup \mathsf{BP} \cup \mathsf{DP}$$

i.e., relating blocking and divergence in the state spaces in the obvious way.

We first unwind the downward simulation conditions. For the moment, we elide the finite internal evolution included with initialisation and after every operation.

Simulation rules on $\widehat{\mathsf{Op}}^{\mathsf{d}}$ are unwound in terms of $\widehat{\widehat{\mathsf{Op}}}$. Specifically, we consider a relation R from AState to CState such that $\widetilde{\mathsf{R}}$ is a downward simulation, i.e.:

$$\begin{array}{l}
\widehat{\mathsf{CInit}}^{\mathsf{d}} \subseteq \widehat{\mathsf{AInit}}^{\mathsf{d}} \fatsemi \widetilde{\mathsf{R}} \\
\widetilde{\mathsf{R}} \fatsemi \widehat{\mathsf{CFin}}^{\mathsf{d}} \subseteq \widehat{\mathsf{AFin}}^{\mathsf{d}} \\
\forall\, i : I \bullet \widetilde{\mathsf{R}} \fatsemi \widehat{\mathsf{COp}_i}^{\mathsf{d}} \subseteq \widehat{\mathsf{AOp}_i}^{\mathsf{d}} \fatsemi \widetilde{\mathsf{R}}
\end{array}$$

The standard relational argument [11, pp. 77–79] shows that the following equations precisely characterise such a downward simulation:

$$\begin{array}{l}
\widehat{\widehat{\mathsf{CInit}}} \subseteq \widehat{\widehat{\mathsf{AInit}}} \fatsemi \mathsf{R} \\
\mathsf{R} \fatsemi \widehat{\widehat{\mathsf{CFin}}} \subseteq \widehat{\widehat{\mathsf{AFin}}} \\
\operatorname{ran}(\operatorname{dom} \widehat{\widehat{\mathsf{AFin}}} \lhd \mathsf{R}) \subseteq \operatorname{dom} \widehat{\widehat{\mathsf{CFin}}} \\
\forall\, i : I \bullet \operatorname{ran}(\operatorname{dom} \widehat{\widehat{\mathsf{AOp}_i}} \lhd \mathsf{R}) \subseteq \operatorname{dom} \widehat{\widehat{\mathsf{COp}_i}} \\
\forall\, i : I \bullet (\operatorname{dom} \widehat{\widehat{\mathsf{AOp}_i}} \lhd \mathsf{R}) \fatsemi \widehat{\widehat{\mathsf{COp}_i}} \subseteq \widehat{\widehat{\mathsf{AOp}_i}} \fatsemi (\mathsf{R} \cup \mathsf{BP})
\end{array}$$

These are unwound to give the equivalent set of conditions on the underlying partial relations. Specifically, they become:

$$\begin{array}{l}
\mathsf{CInit} \uparrow \Rightarrow \mathsf{AInit} \uparrow \\
\neg \mathsf{AInit} \uparrow \Rightarrow \mathsf{CInit} \subseteq \mathsf{AInit} \fatsemi \mathsf{R} \\
\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin} \\
\forall\, i : I \bullet \operatorname{ran}(\neg \operatorname{div} \mathsf{AOp}_i \lhd \mathsf{R}) \subseteq \neg \operatorname{div} \mathsf{COp}_i \\
\forall\, i : I \bullet \operatorname{ran}(\neg \operatorname{div} \mathsf{AOp}_i \cap \operatorname{dom} \mathsf{AOp}_i \lhd \mathsf{R}) \subseteq \operatorname{dom} \mathsf{COp}_i \\
\forall\, i : I \bullet (\neg \operatorname{div} \mathsf{AOp}_i \lhd \mathsf{R}) \fatsemi \mathsf{COp}_i \subseteq \mathsf{AOp}_i \fatsemi \mathsf{R}
\end{array}$$

We now translate these relational conditions into conditions on the Z representation of the abstract data type. Finite internal evolution after initialisation and operations is still elided.

$$
\begin{array}{l}
\mathit{CInit} \uparrow \Rightarrow \mathit{AInit} \uparrow \\
\neg \mathit{AInit} \uparrow \Rightarrow \forall \mathit{CState}' \bullet \mathit{CInit} \Rightarrow \exists \mathit{AState}' \bullet \mathit{AInit} \land R' \\
\forall i : I \bullet \forall \mathit{AState};\ \mathit{CState} \bullet \mathrm{div}\, \mathit{COp}_i \land R \Rightarrow \mathrm{div}\, \mathit{AOp}_i \\
\forall i : I \bullet \forall \mathit{AState};\ \mathit{CState} \bullet \mathrm{pre}\, \mathit{AOp}_i \land R \Rightarrow (\mathrm{pre}\, \mathit{COp}_i \lor \mathrm{div}\, \mathit{AOp}_i) \\
\forall i : I \bullet \forall \mathit{AState};\ \mathit{CState};\ \mathit{CState}' \bullet \\
\qquad\qquad\qquad \neg\, \mathrm{div}\, \mathit{AOp}_i \land R \land \mathit{COp}_i \Rightarrow \exists \mathit{AState}' \bullet R' \land \mathit{AOp}_i
\end{array}
$$

Thus these are the Z schema calculus downward simulation rules for a blocking semantics without consideration of input/output. Making the finite internal evolution explicit once again gives:

$$
\begin{array}{l}
\mathit{CInit} \uparrow \Rightarrow \mathit{AInit} \uparrow \\
\neg \mathit{AInit} \uparrow \Rightarrow \forall \mathit{CState}' \bullet \mathit{CInit} \fatsemi \tau_C^* \Rightarrow \exists \mathit{AState}' \bullet \mathit{AInit} \fatsemi \tau_A^* \land R' \\
\forall i : I \bullet \forall \mathit{AState};\ \mathit{CState} \bullet \mathrm{div}\, \mathit{COp}_i \land R \Rightarrow \mathrm{div}\, \mathit{AOp}_i \\
\forall i : I \bullet \forall \mathit{AState};\ \mathit{CState} \bullet \mathrm{pre}\, \mathit{AOp}_i \land R \Rightarrow (\mathrm{pre}\, \mathit{COp}_i \lor \mathrm{div}\, \mathit{AOp}_i) \\
\forall i : I \bullet \forall \mathit{AState};\ \mathit{CState};\ \mathit{CState}' \bullet \\
\qquad\qquad \neg\, \mathrm{div}\, \mathit{AOp}_i \land R \land \mathit{COp}_i \fatsemi \tau_C^* \Rightarrow \exists \mathit{AState}' \bullet R' \land \mathit{AOp}_i \fatsemi \tau_A^*
\end{array}
$$

(Note that $\mathrm{pre}(\mathit{Op} \fatsemi \tau^*) \equiv \mathrm{pre}\, \mathit{Op}$, and analogously for div.)

4.2.2 Upward simulations

We now perform a similar unwinding on the upward simulation rules. Specifically, we consider T from CState to AState satisfying

$$
\begin{array}{l}
\widehat{\mathsf{CInit}}^{\mathsf{d}} \fatsemi \widetilde{\mathsf{T}} \subseteq \widehat{\mathsf{AInit}}^{\mathsf{d}} \\
\widehat{\mathsf{CFin}}^{\mathsf{d}} \subseteq \widetilde{\mathsf{T}} \fatsemi \widehat{\mathsf{AFin}}^{\mathsf{d}} \\
\forall i : I \bullet \widehat{\mathsf{COp}_i}^{\mathsf{d}} \fatsemi \widetilde{\mathsf{T}} \subseteq \widetilde{\mathsf{T}} \fatsemi \widehat{\mathsf{AOp}_i}^{\mathsf{d}}
\end{array}
$$

and elide finite internal evolution for the moment.

The standard relational argument [11, pp. 79–80] then shows that the following equations precisely characterise such an upward simulation:

$$
\begin{array}{l}
\widehat{\widehat{\mathsf{CInit}}} \fatsemi \mathsf{T} \subseteq \widehat{\widehat{\mathsf{AInit}}} \\
\widehat{\widehat{\mathsf{CFin}}} \subseteq \mathsf{T} \fatsemi \widehat{\widehat{\mathsf{AFin}}} \\
\forall \mathsf{c} : \mathsf{CSt}\underline{\mathsf{ate} \bullet \mathsf{T}(\!|\ \{\mathsf{c}\}}\ |\!) \subseteq \mathrm{dom}\, \widehat{\widehat{\mathsf{AFin}}} \Rightarrow \mathsf{c} \in \mathrm{dom}\, \widehat{\widehat{\mathsf{CFin}}} \\
\forall i : I \bullet \mathrm{dom}\, \widehat{\widehat{\mathsf{COp}_i}} \subseteq \mathrm{dom}(\mathsf{T} \rhd \mathrm{dom}\, \widehat{\widehat{\mathsf{AOp}_i}}) \\
\forall i : I \bullet \mathrm{dom}(\mathsf{T} \rhd \mathrm{dom}\, \widehat{\widehat{\mathsf{AOp}_i}}) \lhd \widehat{\widehat{\mathsf{COp}_i}} \fatsemi \mathsf{T} \subseteq (\mathsf{T} \cup \mathsf{BP}) \fatsemi \widehat{\widehat{\mathsf{AOp}_i}}
\end{array}
$$

Also taking account of finite internal evolution, we translate these to:

$$
\begin{array}{l}
\mathit{CInit} \uparrow \Rightarrow \mathit{AInit} \uparrow \\
\neg \mathit{AInit} \uparrow \Rightarrow \forall\, \mathit{AState}';\ \mathit{CState}' \bullet \mathit{CInit} \fatsemi \tau_C^* \wedge T' \Rightarrow \mathit{AInit} \fatsemi \tau_A^* \\
\forall\, \mathit{CState} \bullet \exists\, \mathit{AState} \bullet T \\
\forall\, i : I \bullet \forall\, \mathit{CState} \bullet \exists\, \mathit{AState} \bullet T \wedge (\mathrm{div}\, \mathit{COp}_i \Rightarrow \mathrm{div}\, \mathit{AOp}_i) \\
\forall\, \mathit{CState} \bullet \exists\, \mathit{AState} \bullet \forall\, i : I \bullet T \wedge (\mathrm{pre}\, \mathit{AOp}_i \Rightarrow (\mathrm{pre}\, \mathit{COp}_i \vee \mathrm{div}\, \mathit{AOp}_i)) \\
\forall\, i : I \bullet \forall\, \mathit{AState}';\ \mathit{CState};\ \mathit{CState}' \bullet \\
\qquad (\mathit{COp}_i \fatsemi \tau_C^* \wedge T') \Rightarrow (\exists\, \mathit{AState} \bullet T \wedge (\mathrm{pre}\, \mathit{AOp}_i \Rightarrow \mathit{AOp}_i \fatsemi \tau_A^*))
\end{array}
$$

5 Conclusions

This paper has been concerned with a relational view of process algebraic refinement. After surveying the current results in the area, we have concentrated on deriving a correspondence between failures-divergences refinement and relational data refinement, in particular in the presence of internal evolution. A precise characterisation of the generalised finalisation needed has been given, as well as a derivation of the additional simulation conditions that are a consequence of the finalisation used.

These results extend [12] by considering the explicit presence of internal events in the blocking model. Further work in this area includes extending the results to the non-blocking model, as well as the full consideration of input and output and the effect that the extra subtleties introduced have upon the results derived above. This will be the subject of an expanded version of this paper [1].

Acknowledgement

We would like to thank Christie Bolton, Jim Davies, Gerhard Schellhorn, Steve Schneider, Helen Treharne and Heike Wehrheim for discussions on these issues. John Derrick was supported by the Leverhulme Trust via a Research Fellowship for this work.

References

[1] Boiten, E. and J. Derrick, *Relational concurrent refinement with internal operations and outputs* (2006), submitted for publication.

[2] Boiten, E. A. and J. Derrick, *Unifying concurrent and relational refinement*, ENTCS **70** (2002), proceedings REFINE'02, Editors: J. Derrick, E. A. Boiten, J. von Wright and J. C. P. Woodcock.

[3] Bolognesi, T. and E. Brinksma, *Introduction to the ISO Specification Language LOTOS*, Computer Networks and ISDN Systems **14** (1988), pp. 25–59.

[4] Bolton, C., "On the refinement of state-based and event-based models," Ph.D. thesis, University of Oxford (2002).

[5] Bolton, C. and J. Davies, *Refinement in Object-Z and CSP*, in: M. Butler, L. Petre and K. Sere, editors, *Integrated Formal Methods (IFM 2002)*, Lecture Notes in Computer Science **2335** (2002), pp. 225–244.

[6] Bolton, C. and J. Davies, *A Singleton Failures Semantics for Communicating Sequential Processes*, Formal Aspects of Computing **18** (2006), pp. 181–210.

[7] Bolton, C. and G. Lowe, *A hierarchy of failures-based models: Theory and application*, Theoretical Computer Science **330** (2005), pp. 407–438.

[8] Brookes, S. and A. Roscoe, *An improved failures model for communicating processes*, in: S. Brookes, A. Roscoe and G. Winskel, editors, *Seminar on Concurrency*, Lecture Notes in Computer Science **197** (1985), pp. 281–305.

[9] Brookes, S. D., C. A. R. Hoare and A. W. Roscoe, *A theory of communicating sequential processes*, Journal of the ACM **31** (1984), pp. 560–599.

[10] de Roever, W.-P. and K. Engelhardt, "Data Refinement: Model-Oriented Proof Methods and their Comparison," CUP, 1998.

[11] Derrick, J. and E. Boiten, "Refinement in Z and Object-Z: Foundations and Advanced Applications," FACIT, Springer Verlag, 2001.

[12] Derrick, J. and E. Boiten, *Relational concurrent refinement*, Formal Aspects of Computing **15** (2003), pp. 182–214.

[13] Derrick, J. and E. Boiten, *Relational concurrent refinement with internal operations (full version)*, Technical report, University of Sheffield (2006).

[14] Jifeng, H., *Process refinement*, in: J. McDermid, editor, *The Theory and Practice of Refinement* (1989).

[15] Josephs, M. B., *A state-based approach to communicating processes*, Distributed Computing **3** (1988), pp. 9–18.

[16] Milner, R., "Communication and Concurrency," Prentice-Hall, 1989.

[17] Roscoe, A., "The Theory and Practice of Concurrency," International Series in Computer Science, Prentice Hall, 1998.

[18] Schneider, S., *Non-blocking data refinement and traces-divergences semantics* (2006), personal communication.

[19] Spivey, J. M., "The Z Notation: A Reference Manual," International Series in Computer Science, Prentice Hall, 1992, 2nd edition.

[20] van Glabbeek, R. J., *The linear time - branching time spectrum I. The semantics of concrete sequential processes*, in: J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, North-Holland, 2001 pp. 3–99.

[21] Woodcock, J. C. P. and J. Davies, "Using Z: Specification, Refinement, and Proof," Prentice Hall, 1996.

[22] Woodcock, J. C. P. 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*, Lecture Notes in Computer Science **428** (1990).