Under consideration for publication in Formal Aspects of Computing

# Relational Concurrent Refinement Part II: Internal Operations and Outputs

Eerke Boiten[1], John Derrick[2] and Gerhard Schellhorn[3]

[1] Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, UK
[2] Department of Computer Science, University of Sheffield, Sheffield, S1 4DP, UK
[3] Department of Computer Science, Institute of Software Engineering and Programming Languages, University of Augsburg, 86135 Augsburg, Germany.

**Abstract.** Two styles of description arise naturally in formal specification: state-based and behavioural. In state-based notations, a system is characterised by a collection of variables, and their values determine which actions may occur throughout a system history. Behavioural specifications describe the chronologies of actions – interactions between a system and its environment. The exact nature of such interactions is captured in a variety of semantic models with corresponding notions of refinement; refinement in state based systems is based on the semantics of sequential programs and is modelled relationally. Acknowledging that these viewpoints are complementary, substantial research has gone into combining the paradigms.

The purpose of this paper is to do three things. First, 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. Second, we generalise these results by providing a general flexible scheme for incorporating the two main "erroneous" concurrent behaviours: deadlock and divergence, into relational refinement. This is shown to subsume previous characterisations. In doing this we derive relational refinement rules for specifications containing both internal operations and outputs that corresponds to failures-divergences refinement. Third, the theory has been formally specified and verified using the interactive theorem prover KIV.

**Keywords:** Data refinement, Z, simulations, process algebraic semantics, failures-divergences refinement, deadlock, internal operations, outputs, mechanisation, KIV.

---

*Correspondence and offprint requests to*: E.A. Boiten, J. Derrick and G. Schellhorn

## 1. Introduction

Two styles of description arise naturally in formal specification: state-based and behavioural. In state-based notations, such as Z [43], a system is characterised by a collection of variables. The system's operations are defined by their effects on the variables, and conditions on their values determine when operations are applicable.

In behavioural notations, e.g., process algebras such as CSP [27, 36], the allowable chronologies of actions are specified explicitly. Moreover, actions actually represent *inter*actions between a system and its environment. The exact way in which the environment is allowed to interact with the system varies between notations, or to be more precise, between concurrency semantics. Typical semantics are set-based, associating one or more sets with each process, for example traces, refusals, divergences. Refinement is then defined in terms of set inclusions and equalities between the corresponding sets for different processes. For example, in CSP one could use trace refinement, failures refinement or failures-divergences refinement [36]. In CCS, bisimulation is typically used [33], whereas in LOTOS reduction, extension and conformance are defined [5]. A survey of many of the most prominent refinement relations is given in [45].

On the other hand, in state-based systems specifications are considered to define abstract data types (ADTs), consisting of an initialisation, a collection of operations and a finalisation, where a program over an ADT is a sequential composition of these elements. Refinement in this context is taken to be the subset relation over program behaviours, where what is deemed visible (i.e., the domain of the initialisation and the range of the finalisation) is the input/output relation. Thus an ADT $\mathsf{C}$ refines an ADT $\mathsf{A}$ if for every program and sequence of inputs, the outputs that $\mathsf{C}$ produces are outputs that $\mathsf{A}$ could also have produced. This definition of refinement quantifies over program behaviour and *simulations* have become the accepted approach to make verification of refinements tractable [16]. Two different forms of simulations are needed to provide a complete method: downward and upward simulations. Theoretical background is given in [16], and examples of their use in Z are given in [46, 17].

Motivated by both theoretical comparisons of refinement and integrations of specification languages, there has been recent interest in relating these two viewpoints of refinement. That is, in order to understand the nature and structure of refinement, as well as provide a means to combine languages and their development methodologies, it is necessary to understand the correspondence between data refinement and process refinement relations.

The purpose of this paper is three-fold. First, we survey this existing work linking relational models of refinement to their process algebraic counterparts. Second, we extend these results, and thirdly we verify the theory mechanically. In doing so, and as a particular by-product of the theory, we derive simulation rules for relational data refinement of specifications containing both internal operations and outputs.

Work relating the two paradigms includes Josephs [30], He [29], Woodcock and Morgan [47], Bolton and Davies [7, 8], Derrick and Boiten [4, 18] and Schneider [40]. That due to Josephs [30], He [29], Woodcock and Morgan [47] defines a basic correspondence between simulation rules and failures-divergences refinement. The more recent work of Bolton and Davies [7, 8], Derrick and Boiten [4, 18] and Schneider [40] investigates a direct correspondence between the relational model and process semantics, and includes specific consideration of input and output which introduces some subtleties.

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 [40] and Bolton and Davies [7, 8] do the former, whilst Derrick and Boiten the latter [18]. Either way, a process semantics $[\![\mathsf{A}]\!]$ can be given for an ADT $\mathsf{A}$. 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 relational data refinement, and $\sqsubseteq_{ps}$ the refinement relation induced by the given process semantics. Varying X, and to some extent $\sqsubseteq_{data}$, gives different process semantics. In particular, we are interested in the semantic models of CSP, for example, traces-divergences or failures-divergences and how they related to data refinement in a relational model.

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

This particular result is due to Schneider [40]. 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 show [18] what additional observations are needed to induce failures-divergences refinement. The latter derive simulation rules for failures-divergences refinement in the blocking model, and recently [19] included the integration of internal operations into this model. The construction relies on a proof that the standard relational semantics is an abstraction of the concurrency semantics, that is, the former does not distinguish between processes that are considered equal by the latter.

Here we generalise these results by providing a general flexible scheme for incorporating the two main "erroneous" concurrent behaviours: deadlock and divergence, into relational refinement. This is shown to subsume previous results. We are also able to derive downward and upward simulation conditions for specifications containing both internal operations and outputs that correspond to failures-divergences refinement. We highlight the role of divergence due to unbounded internal evolution and the relation between outputs and refusals, and their effect on the simulation conditions.

The paper is structured as follows. In Section 2 we provide the basic definitions and background, and we survey existing work in Section 3. The new relational datatype is defined in Section 4, and refinement rules for it are derived through its embedding in the standard theory. Sections 5 and 6 instantiate this new data type with a number of varieties of data types with internal operations and outputs. We conclude in Section 7 which also discusses the formalisation of the theory in KIV.

## 2. Background

In this section we present the standard refinement theory [17] for abstract data types in a relational setting. We describe the relational model for abstract data types written as total relations over a global state in Section 2.1 and that for partial relations in Section 2.2. Section 2.3 introduces process algebraic variants of refinement that are relevant to our discussion. Section 2.4 shows how the relational theory can be applied to a specification language such as Z, and Section 2.5 studies the global state in more detail.

### 2.1. The standard relational model

Relational abstract data types are centred around a hidden local state $\mathsf{State}$. However, their semantics are defined in terms of observations on a visible "global" state $\mathsf{G}$. These observations are induced by *programs* which are characterised by (sequences of) invocations of the ADT's operations. The *initialisation* of the program takes a global state to a local state, on which the operations act, a *finalisation* translates back from local to global. The semantics of a program is then a relation on the global state: an initialisation, followed by operations on the local state, followed by a finalisation.

In order to distinguish between relational formulations (which use Z as a meta-language) and expressions in terms of Z schemas etc., we introduce the convention that expressions and identifiers in the world of relational data types are typeset in a sans serif font.

**Definition 1 (Basic data type; Total data type; Program controlled ADT).**
A *basic data type* is a quadruple $(\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i \in I}, \mathsf{Fin})$. The operations $\{\mathsf{Op}_i\}$, indexed by $i \in I$, are relations on the set $\mathsf{State}$; $\mathsf{Init}$ is a total relation from $\mathsf{G}$ to $\mathsf{State}$; $\mathsf{Fin}$ is a relation from $\mathsf{State}$ to $\mathsf{G}$. If all operations $\{\mathsf{Op}_i\}$ and $\mathsf{Fin}$ are total relations, we call it a *total data type*.

A basic ADT with initialisation $\mathsf{Init}$ and global state $\mathsf{G}$ is *program controlled* if the initial global state is irrelevant for initialisation, i.e., $\mathsf{Init} = \mathsf{G} \times \operatorname{ran} \mathsf{Init}$. □

All ADTs in this paper will be program controlled, that is, the global state before initialisation is irrelevant. This ensures that the output of an ADT run is fully determined by the program, not by any other global state information. This does not mean that initialisation is irrelevant: it still describes in which local state the ADT might start, but this choice is not influenced by outside information.

**Definition 2 (Program; Data refinement).**
A *program* over a data type $\mathsf{D} = (\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i \in I}, \mathsf{Fin})$ is a sequence over the index set $I$, which is identified with the sequential composition of the corresponding operations. For a program $\mathsf{p}$, the corresponding *complete program* for $\mathsf{p}$ in $\mathsf{D}$, denoted $\mathsf{p_D}$, is the relational composition $\mathsf{Init} \mathbin{;} \mathsf{p} \mathbin{;} \mathsf{Fin}$. For example, if $\mathsf{p} = \langle \mathsf{p_1}, ..., \mathsf{p_n} \rangle$ then $\mathsf{p_D} = \mathsf{Init} \mathbin{;} \mathsf{Op_{p_1}} \mathbin{;} ... \mathbin{;} \mathsf{Op_{p_n}} \mathbin{;} \mathsf{Fin}$.

For total data types $\mathsf{A}$ and $\mathsf{C}$, $\mathsf{C}$ *refines* $\mathsf{A}$, denoted $\mathsf{A} \sqsubseteq_{data} \mathsf{C}$, iff for each finite sequence $\mathsf{p}$ over $I$, we have $\mathsf{p_C} \subseteq \mathsf{p_A}$. □

As usual we assume that the data types are *conformal*, i.e., they use the same index set for the operations. Then *downward* and *upward* simulations form a sound and jointly complete proof method for verifying refinements [25, 16]. In a simulation a step-by-step comparison is made of each operation in the data types, and to do so the concrete and abstract states are related by a retrieve relation.

**Definition 3 (Downward simulation).**
Assume total data types $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in I}, \mathsf{AFin})$ and $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in I}, \mathsf{CFin})$. A *downward* simulation is a relation $\mathsf{R}$ between $\mathsf{AState}$ and $\mathsf{CState}$ satisfying

$$\begin{array}{l} \mathsf{CInit} \subseteq \mathsf{AInit} \mathbin{;} \mathsf{R} \\ \mathsf{R} \mathbin{;} \mathsf{CFin} \subseteq \mathsf{AFin} \\ \forall\, i : I \bullet \mathsf{R} \mathbin{;} \mathsf{COp}_i \subseteq \mathsf{AOp}_i \mathbin{;} \mathsf{R} \end{array}$$

If such a simulation exists, we also say that $\mathsf{C}$ is a downward simulation of $\mathsf{A}$ and similarly for the corresponding operations of $\mathsf{A}$ and $\mathsf{C}$. □

**Definition 4 (Upward simulation).**
For total data types $\mathsf{A}$ and $\mathsf{C}$ as above, an *upward* simulation is a relation $\mathsf{T}$ between $\mathsf{CState}$ and $\mathsf{AState}$ such that

$$\begin{array}{l} \mathsf{CInit} \mathbin{;} \mathsf{T} \subseteq \mathsf{AInit} \\ \mathsf{CFin} \subseteq \mathsf{T} \mathbin{;} \mathsf{AFin} \\ \forall\, i : I \bullet \mathsf{COp}_i \mathbin{;} \mathsf{T} \subseteq \mathsf{T} \mathbin{;} \mathsf{AOp}_i \end{array}$$ □

## 2.2. Partial Relations

Definition 2 and the simulations defined above provide a data refinement methodology for *total* ADTs only. However, in general, operations (for example in Z) may be *partial* relations. The domain of an operation is the collection of before-states where it is required to deliver a well-defined result; from states outside the domain, the operation may either be *forbidden* or its result *unprescribed*. In the former case, we call this the *blocking* approach, and the domain acts as a *guard*; in the *non-blocking* approach, it acts as a *pre-condition*.

Consequently, there are two possibilities for a refinement theory for partial relations deriving from the total relations theory described above. They require an embedding of partial relations into total relations, so-called "totalisations". This is done by adding a $\perp$ value to the state space to represent "erroneous" behaviour, where we let $\mathsf{State}_\perp = \mathsf{State} \cup \{\perp\}$ for some $\perp \notin \mathsf{State}$. Two ways of defining a totalisation are the following[1].

**Definition 5 (Blocking and Non-Blocking Totalisation).**
For a partial relation $\mathsf{Op}$ on $\mathsf{State}$, its totalisation is a total relation on $\mathsf{State}_\perp$, defined in the non-blocking model by

$$\widehat{\mathsf{Op}}^{\mathsf{nb}} == \mathsf{Op} \cup \{\mathsf{x}, \mathsf{y} : \mathsf{State}_\perp \mid \mathsf{x} \notin \mathrm{dom}\, \mathsf{Op} \bullet (\mathsf{x}, \mathsf{y})\}$$

or in the blocking model by

$$\widehat{\mathsf{Op}}^{\mathsf{b}} == \mathsf{Op} \cup \{\mathsf{x} : \mathsf{State}_\perp \mid \mathsf{x} \notin \mathrm{dom}\, \mathsf{Op} \bullet (\mathsf{x}, \perp)\} \qquad \square$$

The simulation rules for partial operations are derived by applying the simulation rules to the totalised relations, and then eliminating $\perp$ from the resulting conditions. For the detailed derivations and the resulting simulation rules on partial relations, see [17] – we will derive a single scheme generalising both the blocking and non-blocking model below.

## 2.3. 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.

Here we will consider semantic models of CSP, of which there are a number, each of which induces its own refinement relation. These refinement relations are closely related to those in other process algebras. However, the CSP models take a particular approach with respect to divergence and it should be noted that other process algebraic models sometimes differ in this respect, see for example, [31] and [44] and the comparison given in [12] between CSP, CCS and LOTOS. We will assume this approach to divergence in our treatment of both process refinement and relational refinement, and this has particular consequences when we look at inclusion of internal events in our framework.

**Failures-divergences semantics** The standard semantics of CSP is the failures-divergences semantics developed in [13, 14, 36]. 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. Unguarded recursion also leads to divergences.

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

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

[1] There are others, which are discussed in [21] and in terms of the terminology defined there the non-blocking totalisation we give is an *unstrict lifting*, and the blocking one is a strict lifting. Deutsch and Henson provide a careful examination of the possible ways to totalise and embed the element $\perp$ into a total data type.

such that a number of properties hold. Properties $F1$ and $F2$ capture the requirement that the sequences of events a process can undergo form a non-empty, prefix-closed set. Property $F3$ states that if a process can refuse all events in a set $X$ then it can refuse all events in any subset of $X$. Property $F4$ states that a process can refuse any event which cannot occur as the next event.

$$(\langle\rangle, \varnothing) \in \mathcal{F} \quad (F1)$$
$$(t_1 \frown t_2, \varnothing) \in \mathcal{F} \Rightarrow (t_1, \varnothing) \in \mathcal{F} \quad (F2)$$
$$(t, X) \in \mathcal{F} \wedge Y \subseteq X \Rightarrow (t, Y) \in \mathcal{F} \quad (F3)$$
$$(t, X) \in \mathcal{F} \wedge (\forall e \in Y \bullet (t \frown \langle e\rangle, \varnothing) \notin \mathcal{F}) \Rightarrow (t, X \cup Y) \in \mathcal{F} \quad (F4)$$

The divergences of a process with alphabet $A$ and failures $\mathcal{F}$ are a set $\mathcal{D} \subseteq A^*$ such that:

$$\mathcal{D} \subseteq \operatorname{dom} \mathcal{F} \quad (D1)$$
$$t_1 \in \mathcal{D} \wedge t_2 \in A^* \Rightarrow t_1 \frown t_2 \in \mathcal{D} \quad (D2)$$
$$t \in \mathcal{D} \wedge X \subseteq A \Rightarrow (t, X) \in \mathcal{F} \quad (D3)$$

The first property states that a divergence is a sequence of events. Properties $D2$ and $D3$ 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*[2].

The failures-divergences semantics induces a refinement ordering defined in terms of failures and divergences [14]. A process $Q$ is a refinement of a process $P$, denoted $P \sqsubseteq_{fd} Q$, iff

$$\mathcal{F}(Q) \subseteq \mathcal{F}(P) \text{ and } \mathcal{D}(Q) \subseteq \mathcal{D}(P)$$

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

$$\mathcal{T}(Q) \subseteq \mathcal{T}(P) \text{ and } \mathcal{D}(Q) \subseteq \mathcal{D}(P)$$

**Singleton failures semantics** The singleton failures semantics for CSP was used by Bolton [6] (and published in [7, 8]) 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:

$$\mathcal{S}(P) = \mathcal{F}(P) \cap (A^* \times \mathbb{P}_1 A)$$

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

[2] The assumptions made here by the CSP models are not necessarily present in other process algebraic semantics.

(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 [45] and later in [11] (which builds on [10]).

### 2.4. Refinement in Z

In Section 2.2 we described how partial relations are totalised in order that the relational theory of data refinement could be defined for a language such as Z. To complete the picture we provide a relational interpretation for Z specifications, that is, provide a semantics in terms of partial relations.

A Z specification can be thought of as a data type, defined as a tuple $(State, Init, \{Op_i\}_{i \in I})$. The operations $Op_i$ are given in terms of (the variables of) $State$ (its before-state) and $State'$ (its after-state). The initialisation is expressed in terms of an after-state $State'$. In addition, operations consume inputs and produce outputs. The standard solution [46, 17] for embedding a Z specification into a relational model is as follows.

**State, Initialisation and Finalisation** The inputs and outputs are observable, so they are added to the global state, and because we require that every operation is embedded into a homogeneous relation, the local state space contains a representation of the Z state together with the sequence of inputs and outputs:

$$\begin{array}{l} \mathsf{G} == \operatorname{seq} Input \times \operatorname{seq} Output \\ \mathsf{State} == \operatorname{seq} Input \times \operatorname{seq} Output \times State \end{array}$$

The initialisation transfers the sequence of inputs from the global state to the local state, and picks an initial local ADT state that satisfies the ADT's initialisation, and the finalisation makes visible the outputs produced by the program.

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

**Operations** An operation $Op$ is modelled as a relation which consumes inputs and produces outputs:

$$\mathsf{Op}_i == \{Op_i;\ is : \operatorname{seq} Input;\ os : \operatorname{seq} Output \bullet (\langle \theta Input \rangle \frown is, os, \theta State) \mapsto (is, os \frown \langle \theta Output \rangle, \theta State')\}$$

**Retrieve relations** If $R$ is a (downward) retrieve relation between $AState$ and $CState$, this is modelled as

$$\mathsf{R} == \{R;\ is : \operatorname{seq} Input;\ os : \operatorname{seq} Output \bullet (is, os, \theta AState) \mapsto (is, os, \theta CState)\}$$

and analogously for upward simulation.

In a context where there is no input or output, the global state contains no information and is a one point domain, i.e., $\mathsf{G} == \{*\}$, and the local state is $\mathsf{State} == State$. In such a context the other components of the embedding collapse to the following:

$$\begin{array}{l} \mathsf{Init} == \{Init \bullet * \mapsto \theta State'\} \\ \mathsf{Op} == \{Op \bullet \theta State \mapsto \theta State'\} \\ \mathsf{Fin} == \{(\theta State, *)\} \\ \mathsf{R} == \{R \bullet \theta AState \mapsto \theta CState\} \end{array}$$

Given these embeddings, we can translate the relational refinement conditions of downward and upward simulations into simulations conditions for Z ADTs. This is straightforward, the only point of note is that the conditions on the finalisation are always satisfied in this Z interpretation.

### 2.5. Observations: Inputs, Outputs and Refusals

A critical decision in developing a refinement theory is: what observations can be made on an ADT? The set of valid observations determines the global state – or, in principle, the aspect that is observed at finalisation. Two types of observations are of particular relevance: outputs and refusals, and we discuss these now.

#### *2.5.1. Inputs and outputs*

The standard construction described above embedded a sequences of inputs and outputs into the local and global states. Here, however, without any loss of generality, we can do away with the input sequence since any quantification over inputs comes hand in hand with the same quantification over the index set $I$. That is, it would make no difference for inputs in operations to appear as a shorthand for an increased alphabet of the ADT[3] – as is commonly the interpretation of inputs in process algebras. Thus, we avoid mentioning inputs from this point. In Z versions of the rules, wherever it says $\forall\, i : I$ we may mentally insert $\forall\, \mathit{Input}$ as well. We do not rely on the conventional assumption that $I$ is finite, and so are not requiring inputs to be from a finite domain either.

The outputs, however, do need an explicit representation in the relational embedding. The results of embeddings where the outputs are part of the global state, and initialisation and finalisation perform copying of them between global and local state are characterised by the following definition. The final two conditions state that every operation adds a single output to the output sequence and copies the rest, and that the previous outputs have no influence on the effect of this operation in general, including the new output value.

**Definition 6 (Output embedding).**
A basic ADT $(\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i\in I}, \mathsf{Fin})$ with global state $\mathsf{G}$ is said to be an *output embedding* iff types $\mathsf{GB}, \mathit{Output}, \mathsf{StateB}$ exist such that

$$
\begin{array}{l}
\mathsf{G} = \mathsf{GB} \times \mathrm{seq}\, \mathit{Output} \\
\mathsf{State} = \mathsf{StateB} \times \mathrm{seq}\, \mathit{Output} \\
\forall\, i, \mathsf{gs}, \mathsf{ls}, \mathsf{ls2}, \mathsf{os}, \mathsf{os2}, \mathsf{out} \bullet \\
\qquad ((\mathsf{gs}, \mathsf{os}), (\mathsf{ls}, \mathsf{os2})) \in \mathsf{Init} \Rightarrow \mathsf{os2} = \langle\rangle \\
\qquad ((\mathsf{ls}, \mathsf{os}), (\mathsf{gs}, \mathsf{os2})) \in \mathsf{Fin} \Rightarrow \mathsf{os2} = \mathsf{os} \\
\qquad ((\mathsf{ls}, \mathsf{os}), (\mathsf{ls2}, \mathsf{os2})) \in \mathsf{Op}_i \Rightarrow \exists\, \mathsf{out} \bullet \mathsf{os2} = \mathsf{os} \frown \langle \mathsf{out} \rangle \\
\qquad ((\mathsf{ls}, \mathsf{os}), (\mathsf{ls2}, \mathsf{os} \frown \langle \mathsf{out} \rangle)) \in \mathsf{Op}_i \Rightarrow ((\mathsf{ls}, \mathsf{os2}), (\mathsf{ls2}, \mathsf{os2} \frown \langle \mathsf{out} \rangle)) \in \mathsf{Op}_i
\end{array}
\qquad \square
$$

#### *2.5.2. Refusals*

We wish to observe traces and refusals[4]. The traces are contained in the standard relational semantics, viz. through the notion of programs. The refusals are not present in the standard embedding given above. Thus to fully encode failures it is necessary to enhance the standard relational theory by adding the observation of refusals at the end of every program. This involves a change only to the global state and the finalisation, as embodied in the following definition.

**Definition 7 (Refusal embedding).**
Consider a basic ADT $(\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i\in I}, \mathsf{Fin})$ with global state $\mathsf{G}$, a particular notion of events $\mathsf{E}$, and

[3] The standard embedding, taking into account ignored outputs at initialisation and ignored inputs at finalisation, effectively leads to a relation of type $\mathrm{seq}\, \mathit{Input} \leftrightarrow \mathrm{seq}\, \mathit{Output}$ for every program, i.e. a semantics of type $\mathrm{seq}\, \mathit{Index} \rightarrow (\mathrm{seq}\, \mathit{Input} \leftrightarrow \mathrm{seq}\, \mathit{Output})$ overall. By not embedding inputs, we are replacing this by $\mathrm{seq}(\mathit{Index} \times \mathit{Input}) \rightarrow \{*\} \leftrightarrow \mathrm{seq}\, \mathit{Output}$, in functional programming terms: uncurrying and zipping.

[4] For discussions about which aspects of a system should or could be observed see [45] and [23].

a refusal relation $\mathsf{Ref} : \mathsf{State} \leftrightarrow \mathbb{P}\,\mathsf{E}$. The ADT is said to be an *refusal embedding* (for $\mathsf{Ref}$) iff it is program controlled, and a type $\mathsf{GB}$ exists such that

$$\mathsf{G} = \mathbb{P}\,\mathsf{E} \times \mathsf{GB}$$
$$\forall\, \mathsf{ls}, \mathsf{r} \bullet (\exists\, \mathsf{gs} \bullet (\mathsf{ls}, (\mathsf{r}, \mathsf{gs})) \in \mathsf{Fin}) \equiv (\mathsf{ls}, \mathsf{r}) \in \mathsf{Ref} \qquad \square$$

As an example, in the context of a Z data type with no input and output, events in $\mathsf{E}$ are simply operation names from $I$, and refusals will be any subset $E$ of the maximal refusals in a given state: $\{i : I \mid \neg\, \mathrm{pre}\, Op_i\}$. Thus an abstract data type $(State, Init, \{Op_i\}_{i \in I})$ in the refusals interpretation is embedded in the relational model as follows. The global state $\mathsf{G}$ is $\mathbb{P}\, I$, finalisation is given by

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

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, see Section 2.4.

Note that a basic ADT can be both an output embedding and a refusal embedding, for example having as the global state $\mathbb{P}\,\mathsf{E} \times \mathrm{seq}\, Output$. Indeed, the presence of outputs has a crucial interaction with the refusals as we discuss now (see also [18]).

In particular, when a system includes a non-deterministic choice of different output values, is the environment able to “select” its choice among these, or is the choice entirely inside the system? The refusals in each model differ slightly, and only the latter option causes refusals through choice of output. These options are called the *angelic* and *demonic* models of outputs, respectively (after [42][5]). In the **Angelic** model the only refusals are the ones arising when an operation is not applicable, there are no refusals due to outputs. In the **Demonic** model [26] a process is, in addition, allowed to refuse all but one of the possible outputs. The difference between the two models is summarised in the following definition.

**Definition 8 (Angelic and Demonic Embeddings).**
An abstract data type $(State, Init, \{Op_i\}_{i \in I})$ with outputs of type $Output$ is embedded in the relational model as follows. The global state is defined by

$$Event == I \times Output$$
$$\mathsf{G} == \mathrm{seq}\, Output \times \mathbb{P}\, Event$$

The embedding of operations is as in Section 2.4; initialisation is extended with an input set of events that is ignored. Finalisation is given by

$$\mathsf{Fin} == \{State;\ os : \mathrm{seq}\, Output;\ E : \mathbb{P}\, Event \mid Fcond \bullet (os, \theta State) \mapsto (os, E)\}$$

with in the angelic model $Fcond = FcondA(\theta State, E)$ where

$$FcondA(s, E) == E \subseteq \{(i, out) \mid \neg\, \exists\, Op_i \bullet \theta State = s \wedge \theta Output = out\}$$

and in the demonic model $Fcond = FcondD(\theta State, E)$ where:

$$\begin{aligned} FcondD(s, E) == E \subseteq \{(i, out) \mid\ & (\neg\, \exists\, Op_i \bullet \theta State = s \wedge \theta Output = out) \vee \\ & (\exists\, Op_i \bullet \theta State = s \wedge \theta Output \neq out \wedge (i, \theta Output) \notin E)\} \end{aligned} \qquad \square$$

We will use this embedding when defining simulation rules that correspond to failures-divergences refinement below.

[5] A more natural naming would be “external” vs. “internal”, in analogy with CSP external and internal choice. However, in this paper the term “internal” is already used in a different context.

## 3. Relating data and process refinement

The previous section outlined the standard relational theory of refinement and 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 [40] |
| Blocking data refinement with deterministic outputs | singleton failures | Bolton and Davies [7, 8] |
| Blocking data refinement | singleton failures of process and input process | Bolton and Davies [7, 8] |
| Blocking data refinement with strengthened applicability but no input/output | failures | Josephs [30] |
| Blocking data refinement with extended finalisations | failures-divergences | Derrick and Boiten [4, 18] |
| Non-blocking data refinement with extended finalisations but no input/output | failures-divergences | Derrick and Boiten [4, 18] |

Both Bolton and Davies and Schneider consider the standard definition of data refinement. The 'extended finalisations' of Derrick and Boiten add conditions to the standard simulation rules, and these are detailed in Section 3.3 below.

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

Inspired by the work by Bolton and Davies [7, 8] discussed below, Schneider [40] 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 as defined above. The notation used is slightly different, but the construction is isomorphic to that given in Section 2.4 above. For such an ADT the translation of an ADT $A$ into a CSP process $process(A)$ is given by

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

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

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

### 3.2. Blocking data refinement and the singleton failures semantics

Bolton in [6] and Bolton and Davies in [7, 8] discuss the relationship between data refinement and the singleton failures semantics [45] 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 $process_b(A)$ is given by the following (using, for uniformity, the notation already introduced):

$$process_b(A) == \sqcap s \in State, (*, s) \in Init \bullet P_A(s)$$

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

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 and with it data refinement corresponds to singleton failures refinement in the process model.

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:

$$inputProcess_b(A) == \sqcap s \in State, (*, s) \in Init \bullet P_A(s)$$

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

As can be seen, this is the same as $process_b$ except that the outputs are unobservable. [8] contains the following result: Blocking data refinement is equivalent to singleton failures refinement of both the process and the input process. That is:

**Theorem 2.** In the blocking model, $A \sqsubseteq_{data} C$ if and only if $process_b(A) \sqsubseteq_{sf} process_b(C)$ and $inputProcess_b(A) \sqsubseteq_{sf} inputProcess_b(C)$. For ADTs with deterministic outputs (or no input/output), this reduces to checking $process_b(A) \sqsubseteq_{sf} process_b(C)$. □

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

$$process_b(A) \sqsubseteq_{fd} process_b(C) \Rightarrow A \sqsubseteq_{data} C$$

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 $process_{nb}$ by

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

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

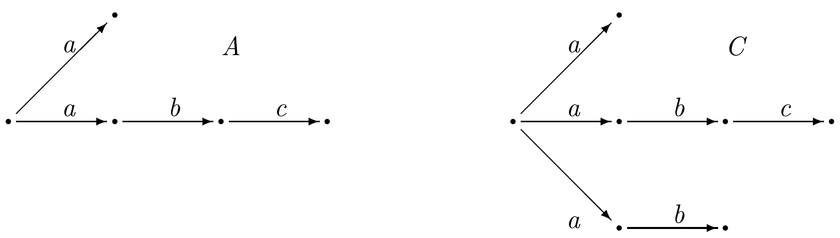


**Fig. 1.** Singleton failures vs. data refinement [34].

where $Chaos == (\sqcap i \in I, in \in Input, out \in Output \bullet AOp_i.in.out \rightarrow Chaos) \sqcap stop$ 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 to the above are derived (i.e., data refinement corresponding to singleton as opposed to failures-divergences refinement). However, this non-blocking translation differs in key aspects from that defined by Schneider, and in particular, the use of *Chaos* and *stop* to model non-termination seems a less natural embedding of non-blocking than using explicit divergence.

However, Reeves and Streader have recently shown [34] that the equivalence given in Theorem 2 only holds if the relational semantics observes the exact point of deadlock, e.g., by producing trivial outputs, and preserving and counting those after deadlock.

Specifically, they discuss the difference between the embedded state $(State \times \text{seq}\, Output)_\perp$ that we use (following [46]) and $(State_\perp \times \text{seq}\, Output)$ which is used by Bolton and Davies [8]. By using the latter, the relational world makes finer distinctions: it is possible to observe how many outputs were produced before the state devolved to $\perp$. This allows to distinguish between, say, a trace *abc* having blocked on *b* or on *c*. The standard relational model does not allow this – in that model we can observe the outputs produced by any prefix of *abc* that does not yet block, but not the additional information of whether blocking immediately after that was possible, if this prefix already allowed blocking. They give the example depicted in Figure 1. The processes $A$ and $C$ in the figure are equivalent in a (standard) relational model with no outputs (in either the blocking or non-blocking model). However, $C$ is not a singleton failures refinement of $A$, adding the failure $(ab, \{c\})$. For relational refinement to correspond to singleton failure refinement, it needs to include a way of observing the exact point of deadlock, for example, by using $(State_\perp \times \text{seq}\, Output)$ as the appropriate embedding.

### 3.3. Defining a correspondence with failures-divergences refinement

Subsequent to the work of Bolton and Davies, Derrick and Boiten in [4, 18] explored what additional conditions were needed on data refinement in order to achieve a correspondence with failures-divergences refinement in a process semantics. To define the appropriate correspondence Derrick and Boiten define a process semantics directly. With no inputs or outputs the traces, failures and divergences can be defined easily in each of the two models.

**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 other ones following divergence. There are no refusals beyond those after a divergence, since before the ADT diverges, 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.

Since refusals are not normally observed in data refinement, it is necessary to observe the refusals directly, and a refusal embedding (see Definition 7) is used. Thus in a context where the data type has no input or output the finalisation used is generalised from being $\{State \bullet \theta State \mapsto *\}$ to becoming $\{State \bullet \theta State \mapsto E\}$.

In defining the process semantics when inputs and outputs are included the only change is to the refusals since the traces and divergences remain the same. When extended to data types with outputs, the effect of adding outputs has consequences for the process semantics, and in particular the refusals of an ADT. Therefore, as detailed in Section 2.5.2, angelic and demonic refusal embeddings as given in Definition 8 are used for the finalisation. The following is proved (irrespective of output model chosen):

**Theorem 3.** In the non-blocking model, for ADTs with no input/output data refinement with extended finalisations corresponds to failures-divergences refinement.

In the blocking model, for ADTs with or without input/output, data refinement with extended finalisations corresponds to failures-divergences refinement. □

**Josephs' construction** For the blocking model without input or output, Josephs in [30] showed that downward and upward simulations with the strengthened applicability condition (1) form a sound and jointly complete method for verifying failures refinement in CSP. Working entirely in the CSP semantics he uses the blocking model in the sense that his CSP processes are projections of ADTs under a semantics where there are no divergences and refusals arise outside an operation's precondition. When the link is made to the relational framework via a corresponding process, the results in [30] are consistent with those in [18].

**Simulation conditions** The downward and upward simulation rules as expressed for partial relations contain conditions on the finalisations. When we use a non-standard finalisation these conditions potentially impose additional constraints on the simulation conditions.

First in a model without input or output. Here, even with a refusal embedding a downward simulation places no further constraints than already present in the standard definition. For an upward simulation the finalisation conditions are:

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

With the refusals embedding the second is always satisfied, however, the first leads to a strengthening of the standard applicability condition from $\forall\, i : I \bullet \forall\, CState \bullet \exists\, AState \bullet\ T \wedge (\mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i)$ to

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

The standard upward simulation applicability condition requires that we consider pairs of abstract and concrete states for each operation. The finalisation condition, on the other hand, requires that for every abstract state we find a *single* concrete state such that the precondition of each abstract operation implies the precondition of its concrete counterpart.

In a model with outputs we distinguish between the demonic and angelic models.

**Demonic model of outputs.** The consequences lie, as before, with the upward simulation conditions, since the downward simulation condition imposed by a refusal embedding is subsumed by the normal applicability and correctness rules.

In the case of upward simulations the finalisation condition leads to an extra condition, which is somewhat complicated involving the need to look at combinations of different operations, whilst considering possible output values individually. This complexity arises from the presence of non-deterministic outputs, and their interaction with the refusal sets. With only deterministic outputs the upward simulation condition is as above (i.e., just involving a strengthened applicability condition). In a model with inputs and outputs, the requirement that $\mathsf{CFin} \subseteq \mathsf{T} \fatsemi \mathsf{AFin}$, simplifies to:

$$\forall\, CState;\ E \bullet FcondD(\theta CState, E) \Rightarrow \exists\, AState \bullet T \wedge FcondD(\theta AState, E) \qquad (2)$$

This forces one to consider different linked abstract states for different maximal concrete refusal sets. In particular, even with just a single operation, it is, in general, necessary to look at different linked states for different output values. In fact, the above condition and its representation below, combine this aspect with the condition derived in the basic construction, which insisted on choosing *the same* linked abstract state for every set of enabled operations.

It is easy to prove that it is sufficient to consider only *maximal* refusal sets in each concrete state. This, and the fact that without outputs there is only one maximal refusal set, allowed to consider only a single linked abstract state. However, in the presence of non-deterministic outputs, multiple maximal refusal sets may exist in each concrete state, each of which may be verified by a different abstract state. See Section 5.6.2 for an example of why the condition is necessary. The condition is, of course, stronger than normal applicability, and additionally implies the totality of $T$.

**Angelic model of outputs.** The effect of the finalisation on the simulation rules is less drastic in the angelic model. Whereas in the demonic model we can reduce non-determinism in the outputs, in the angelic model one cannot. This difference is easily expressed in the simulation rules by using a different precondition operator, defined as $\mathrm{Pre}\, Op == \exists\, State' \bullet Op$, and then using Pre in place of pre in the strengthened applicability conditions.

A complete characterisation of the simulations rules can now be given. For downward simulation, we have the following potential collection of conditions:

**DS.Init** $\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \wedge R'$
**DS.App** $\forall\, CState;\ AState;\ i : I \bullet R \wedge \mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i$
**DS.CorrNonBlock**
$\forall\, i : I;\ Output;\ CState';\ CState;\ AState \bullet \mathrm{pre}\, AOp_i \wedge R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i$
**DS.CorrBlock** $\forall\, i : I;\ Output;\ CState';\ CState;\ AState \bullet R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i$
**DS.FinAng** $\forall\, CState;\ AState;\ i : I;\ Output \bullet R \wedge \mathrm{Pre}\ \ AOp_i \Rightarrow \mathrm{Pre}\ \ COp_i$

Non-blocking data refinement (the standard model and the one corresponding to traces-divergences refinement) requires: **DS.Init**, **DS.App** and **DS.CorrNonBlock**.

Blocking data refinement requires: **DS.Init**, **DS.App** and **DS.CorrBlock**.

For a refusals embedding in the blocking model the following rules are required in the various situations. Each column represents a particular model for (inputs and) outputs; a missing entry indicates a condition dominated by the other conditions in the same column.

| Outputs: | none | demonic | angelic |
|---|---|---|---|
| Init | **DS.Init** | | |
| App | **DS.App** | | - |
| Corr | **DS.CorrBlock** | | |
| Fin | - | | **DS.FinAng** |

For upward simulation, we have the totality of $T$ on $CState$ plus the following set:

**US.Init** $\forall\, CState';\ AState' \bullet T' \wedge CInit \Rightarrow AInit$
**US.AppBlock** $\forall\, i : I;\ Output \bullet \forall\, CState \bullet \exists\, AState \bullet T \wedge \mathrm{pre}\, AOp_i \Rightarrow \mathrm{pre}\, COp_i$

**US.CorrNonBlock**
$\forall\, i : I;\ Output;\ AState';\ CState';\ CState \bullet T' \wedge COp_i \Rightarrow \exists\, AState \bullet T \wedge (\text{pre}\, AOp_i \Rightarrow AOp_i)$

**US.CorrBlock** $\forall\, i : I;\ Output;\ AState';\ CState';\ CState \bullet T' \wedge COp_i \Rightarrow \exists\, AState \bullet T \wedge AOp_i$

**US.FinRef** $\forall\, CState \bullet \exists\, AState \bullet T \wedge \forall\, i : I \bullet \text{pre}\, AOp_i \Rightarrow \text{pre}\, COp_i$

**US.FinDem** $\forall\, CState;\ E \bullet FcondD(\theta CState, E) \Rightarrow \exists\, AState \bullet T \wedge FcondD(\theta AState, E)$

**US.FinAng** $\forall\, CState \bullet \exists\, AState \bullet T \wedge \forall\, i : I;\ Output \bullet \text{Pre}\, AOp_i \Rightarrow \text{Pre}\, COp_i$

Non-blocking data refinement (the standard model and the one corresponding to traces-divergences refinement) requires: **US.Init**, **US.App** and **US.CorrNonBlock**.

Blocking data refinement (corresponding to singleton-failures refinement) requires: **US.Init**, **US.App** and **US.CorrBlock**.

For a refusals embedding in the blocking model the following rules are required in the various situations.

| Outputs: | none | demonic | angelic |
|---|---|---|---|
| Init | **US.Init** | | |
| App | - | | |
| Corr | **US.CorrBlock** | | |
| Fin | **US.FinRef** | **US.FinDem** | **US.FinAng** |

## 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 a few examples.

**Without input/output - non-blocking.** 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 input/output (specifically without output) the process semantics obtained identifies traces-divergences refinement and failures-divergences refinement, that is, $process(A) \sqsubseteq_{td} process(C)$ iff $process(A) \sqsubseteq_{fd} process(C)$. 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.

Consider Figure 2, 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 $e$ it is not the case that any abstract state has

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

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.

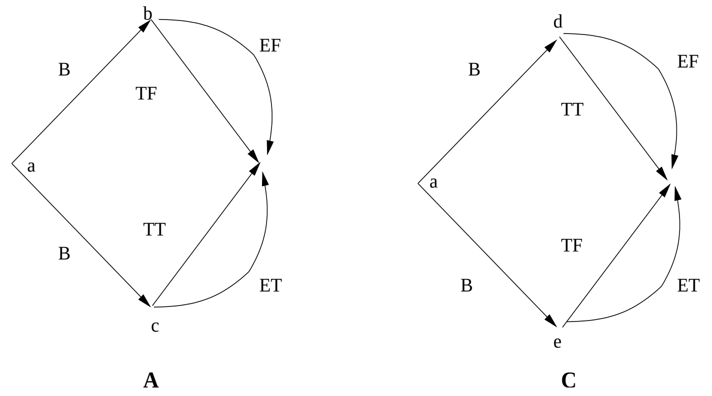


**Fig. 2.** Non-blocking, no input/output = traces-divergences and failures-divergences

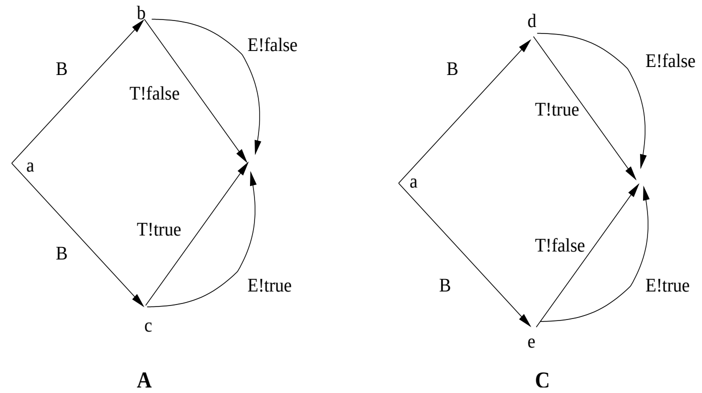


**Fig. 3.** Non-blocking with input/output = traces-divergences but not failures-divergences

**Without input/output - blocking.** 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 $(\langle B\rangle, \{TF, EF\})$ which is not present in $C$, under a singleton failures model in $A$ we just obtain singleton failures $(\langle B\rangle, \{TF\})$, $(\langle B\rangle, \{EF\})$, ... 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.

**With input/output - non-blocking.** When we extend to consider input/output, the presence and modelling of outputs forces a different condition (i.e., different, but related, to the strengthened applicability condition **US.FinRef**) required to recover failures-divergences equivalent.

To see why non-blocking data refinement is not failures-divergences refinement consider Figure 3, where different operations above have been replaced by an operation outputting a different value. Again, in the non-blocking model we have the same traces and same divergences in each specification. The presence of outputs, and non-determinism, causes failures in both specifications. For example, $b$ has as its refusals $\mathbb{P}\{E!true, T!true\}$, whereas $d$ has as its refusals $\mathbb{P}\{E!true, T!false\}$ and $e$ has $\mathbb{P}\{E!false, T!true\}$.

This difference is not visible in the traces-divergences model, and consequently not in program observations made in the relational model. Moving some of the observable information (operation names here) from the first example into the outputs has kept that information observable but not under control of the environment, and it is this information that is captured in the failures.

We have discussed above the conditions that are required in a blocking model. The non-blocking model was not considered in [18], and Section 6.3 derives the extra condition that needs to be enforced in order to recover failures-divergences refinement. We will find that the condition is, of course, similar to the condition **US.FinDem** needed in the blocking model, but that it does not imply the strengthened applicability condition **US.FinRef**. For reasons we discuss later a condition such as **US.FinRef** is not needed in the non-blocking model.

**With input/output - blocking.** Considering the blocking model we already know we need an additional strengthening of applicability (even without input/output) to regain failures-divergences. This example shows that we need the condition **US.FinDem** on refusal sets due to outputs as well. Now the strengthened applicability condition holds - each state $b$, $c$, $d$ and $e$ has operations $E$ and $T$ enabled - so this does not pick up the different refusal information due to the outputs. Thus we need to impose **US.FinDem** to ensure this.

Note that in the case of the blocking model, **US.FinDem** implies **US.FinRef**. To see this, given a concrete state $CS$ and maximal refusal set $E$ in that state, then events are in $E$ if the associated operation is blocked or there exists an alternate output for that operation. For this $E$ we can find an abstract state $AS$ such that $FcondD(A, E)$ holds. Now for this $AS$ consider any operation $Op$, if pre $AOp$ holds in $AS$ but pre $COp$ does not hold in $CS$, then we have violated **US.FinDem**. Hence **US.FinRef** holds.

## 4. A Relational ADT with Divergence and Blocking

In the previous section, we sketched how partial relations would be interpreted in the model of total relation refinement, depending on whether the blocking or non-blocking approach was chosen. However, we have stated previously [32, 3] that the two approaches are not exclusive – and indeed, formalisms such as B [1] have both preconditions and guards. In general, specification formalisms may include both situations that lead to *divergence* and situations that lead to *deadlock*. For example, in our treatment of internal operations in [19] both occur – divergence appears through livelock, and the blocking interpretation is used for operations that are not enabled. The solution chosen there is to apply two totalisations in sequence: one to account for deadlock, and another to account for livelock. What follows here is a generalised reconstruction of that solution, which allows us to derive refinement rules directly for any relational formalism which gives rise to both kinds of errors once the areas of divergence and blocking have been made explicit. In fact, it covers *any* relational formalism modelling at most two kinds of errors, one of which is chaotic, and whose combination satisfies the constraints discussed next.

Indeed, an important consideration is the relative ordering of the two kinds of erroneous behaviours. In particular, we need to decide what observations should be possible when the semantics leads to a non-deterministic choice[6] between any combination of the three behaviours: "normal", "divergent" and "blocking".

First, "divergent" behaviour is usually viewed as "anything might happen", which means that a choice between divergent and normal behaviour should appear as divergence. Consistent with the CSP chaotic interpretation of divergence (e.g., after divergence any refusal is possible), the choice between divergence and blocking should also result in divergence. All in all, this means that there is no observable difference between possible and certain divergence, and that divergence is a zero of non-deterministic choice. The remaining issue is the choice between normal and blocking behaviour. It would be possible, using a model of partial relations (see earlier discussion, and also [16, chapters 8-9]) to take deadlock as a unit of choice, and therefore to not observe possible blocking. Consistent with usual semantics for Z and for CSP, we will

[6] We do not explicitly identify choice operators in our formalism, although availability of multiple operations in a single state is close to external choice, and non-determinism in after-states and choice of outputs in the demonic model are closely related to internal choice.

distinguish possible blocking in our model. These decisions are summarised in the following table, which also informally hints at the set-based model for this which we will introduce formally later. In particular, $\perp$ represents blocking, and $\omega$ represents divergence. In the model, set union acts as the choice operator.

| Choice | normal | divergence | deadlock | poss deadlock | model |
|---|---|---|---|---|---|
| normal | normal | divergence | poss deadlock | poss deadlock | sets not containing $\perp$, $\omega$ |
| divergence | divergence | divergence | divergence | divergence | $\mathsf{State} \cup \{\omega\}$, possibly also $\perp$ |
| deadlock | poss deadlock | divergence | deadlock | poss deadlock | $\{\perp\}$ |
| poss deadlock | poss deadlock | divergence | poss deadlock | poss deadlock | sets containing $\perp$ but not $\omega$ |

With these considerations in mind, we now define a relational data type with partial operations allowing for both divergence and blocking. Its reduction is the basic data type obtained by removing all blocking and divergence information.

**Definition 9 (Process data type; Reduction).**
A *process data type* is a quadruple

$$(\mathsf{State}, \mathsf{Inits}, \{\mathsf{Op}_i\}_{i \in I}, \mathsf{Fin})$$

where $\mathsf{Inits}$ is a subset of $\mathsf{State}$; every operation $\{\mathsf{Op}_i\}$ is a triple $(\mathsf{N}, \mathsf{B}, \mathsf{D})$ such that $\mathrm{dom}\,\mathsf{N}$, $\mathsf{D}$ and $\mathsf{B}$ form a partition of $\mathsf{State}$; $\mathsf{Fin}$ is a relation from $\mathsf{State}$ to $\mathsf{G}$.

Its *reduction* is the basic data type $(\mathsf{State}, \mathsf{G} \times \mathsf{Inits}, \{\mathsf{N}_i\}_{i \in I}, \mathsf{Fin})$ □

In an operation $\mathsf{Op} = (\mathsf{N}, \mathsf{B}, \mathsf{D})$ the relation $\mathsf{N}$ represents the operation's normal effect; the sets $\mathsf{B}$ and $\mathsf{D}$ represent states where the operation would lead to blocking and divergence, respectively. The three sets forming a partition excludes certain situations, such as miracles and possible (as opposed to certain) deadlock from a given state, and ensures that it can be represented by a *total* data type. Possible deadlock still occurs, however, whenever a program leads to multiple states, some but not all of which are deadlocked.

The blocking and non-blocking approaches to operations are, of course, special cases of process data types, in particular:

- the blocking operation $\mathsf{Op}$ is represented by $(\mathsf{Op}, \overline{\mathrm{dom}\,\mathsf{Op}}, \varnothing)$, i.e., it never diverges, and blocks in the complement of the operation's domain;
- the non-blocking operation $\mathsf{Op}$ is represented by $(\mathsf{Op}, \varnothing, \overline{\mathrm{dom}\,\mathsf{Op}})$, i.e., it diverges in the complement of the operation's domain, but never blocks.

The definition of a process data type is not intended as a new and wonderful relational specification mechanism, to compete with similar approaches such as [2, 22, 24, 28]. It is solely an intermediate formalism that most partial relation frameworks can be embedded into. In turn, it is embedded into the total relations framework in order to define its simulation rules once and for all. The embedding is given below. In effect, it fixes the semantics of a process data type.

In the embeddings we will use state spaces enhanced with special values defined below. For simplicity, we assume in the rest of this paper that $\perp$ (representing blocking), $\omega$ (representing divergence), and $\mathsf{no}$ are different values not already contained in any local or global state space of interest. The impossibility of making an observation in a final state is encoded in $\mathsf{no}$, and this is added to the global state only. The embedding (and semantics) of a process data type is then defined as follows.

**Definition 10 (Enhanced state; Embedding of a process data type).**
For any set $\mathsf{State}$, let

$$\mathsf{State}_{\perp,\omega,\mathsf{no}} == \mathsf{State} \cup \{\perp, \omega, \mathsf{no}\}$$

and similarly for sets subscripted with subsets of these special values.

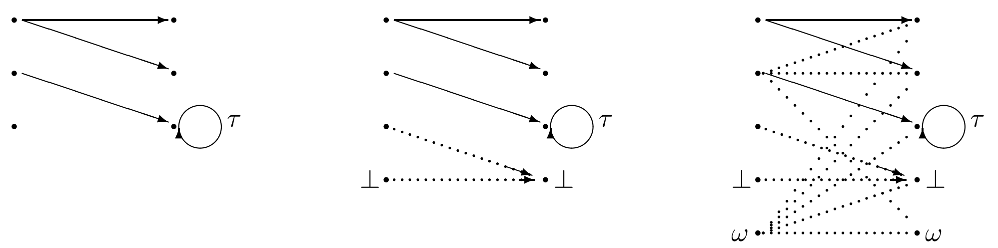


**Fig. 4.** The original $\mathsf{Op}$, and a divergent after-state; with $\mathsf{B}_\perp \times \{\perp\}$ added; finally also with $\mathsf{D}_\omega \times \mathsf{State}_\omega \cup \{(\omega, \perp)\}$.

A process data type $(\mathsf{State}, \mathsf{Inits}, \{(\mathsf{N}_i, \mathsf{B}_i, \mathsf{D}_i)\}_{i \in I}, \mathsf{Fin})$ with global state $\mathsf{G}$ is embedded into the total data type $(\mathsf{State}_{\perp,\omega}, \mathsf{Init}, \{[\![\mathsf{Op}]\!]_\mathsf{i}\}_{i \in I}, [\![\mathsf{Fin}]\!])$ where

$$\mathsf{Init} == \mathsf{G}_{\perp,\omega,\mathsf{no}} \times \mathsf{Inits}$$
$$[\![(\mathsf{N}, \mathsf{B}, \mathsf{D})]\!] == \mathsf{N} \cup (\mathsf{B}_{\perp,\omega} \times \{\perp\}) \cup (\mathsf{D}_\omega \times \mathsf{State}_\omega)$$
$$[\![\mathsf{Fin}]\!] == \mathsf{Fin} \cup (\overline{\mathrm{dom}\,\mathsf{Fin}} \times \{\mathsf{no}\}) \cup \{(\perp, \perp)\} \cup \{\omega\} \times \mathsf{G}_{\omega,\mathsf{no}} \qquad \square$$

As a process data type has a set of initial states rather than an initialisation relation, the embedding's initialisation relates every global state to all such states. The normal effect of an operation is part of the embedded one. In addition, every blocking state including $\perp$ is related to $\perp$, every state where the operation diverges to every state including $\omega$, and divergent state $\omega$ is linked to all states even including $\perp$. Finalisation makes both blocking and divergence visible globally. Figure 4 illustrates this operation embedding.

Refinement (i.e., downward and upward simulation) of process data types is derived by embedding them into total data types[7], applying the simulation rules for total data types, and then eliminating $\perp$, $\omega$, and $\mathsf{no}$ from the resulting rules – i.e., expressing them in terms of the process data types only.

## 4.1. Downward simulation for process data types

Consider the process data types $(\mathsf{AState}, \mathsf{AInits}, \{\mathsf{AOp}_i\}_{i \in I}, \mathsf{AFin})$ and $(\mathsf{CState}, \mathsf{CInits}, \{\mathsf{COp}_i\}_{i \in I}, \mathsf{CFin})$, both with global state $\mathsf{G}$, and a candidate downward simulation relation $\mathsf{R}$ between $\mathsf{AState}$ and $\mathsf{CState}$. We also embed the simulation relation, in order to relate abstract and concrete blocking and divergence correctly:

$$[\![\mathsf{R}]\!] == \mathsf{R} \cup \{(\perp, \perp)\} \cup \{\omega\} \times \mathsf{CState}_\omega$$

**Initialisation**

Applying the initialisation condition on the embedded data types with the embedded simulation relation $[\![\mathsf{R}]\!]$ we calculate:

$$\mathsf{CInit} \subseteq \mathsf{AInit} \mathbin{⨾} [\![\mathsf{R}]\!]$$
$$\equiv \{ \text{ definition of embeddings } \}$$
$$\mathsf{G}_{\perp,\omega} \times \mathsf{CInits} \subseteq (\mathsf{G}_{\perp,\omega} \times \mathsf{AInits}) \mathbin{⨾} [\![\mathsf{R}]\!]$$
$$\equiv \{ \text{ definition of } [\![\mathsf{R}]\!] \}$$
$$\mathsf{CInits} \subseteq \mathrm{ran}(\mathsf{AInits} \lhd \mathsf{R})$$

[7] The embedding of the initialisation is unusual, in the sense that it is non-strict in the erroneous behaviours $\perp$, $\omega$, and $\mathsf{no}$. This choice was made to ensure the ADT is program controlled; if we were to consider sequential composition of ADTs, strictness would be required.

**Finalisation**

This derivation is shown in detail; similar steps in later derivations (revolving around distribution of $\cup$ over $\fatsemi$, and subsequent simplifications) will be contracted.

$$\llbracket \mathsf{R} \rrbracket \fatsemi \llbracket \mathsf{CFin} \rrbracket \subseteq \llbracket \mathsf{AFin} \rrbracket$$
$$\equiv \{ \text{ embeddings } \}$$
$$(\mathsf{R} \cup \{(\bot,\bot)\} \cup \{\omega\} \times \mathsf{CState}_\omega) \fatsemi (\mathsf{CFin} \cup \{(\bot,\bot)\} \cup \{\omega\} \times \mathsf{G}_\omega \cup (\overline{\mathrm{dom}\,\mathsf{CFin}} \times \{\mathsf{no}\}))$$
$$\subseteq$$
$$\mathsf{AFin} \cup \{(\bot,\bot)\} \cup \{\omega\} \times \mathsf{G}_\omega \cup (\overline{\mathrm{dom}\,\mathsf{AFin}} \times \{\mathsf{no}\})$$
$$\equiv \{ \text{ distribution of } \cup \text{ over } \fatsemi \}$$
$$\mathsf{R} \fatsemi \mathsf{CFin} \;\cup\; \mathsf{R} \fatsemi (\overline{\mathrm{dom}\,\mathsf{CFin}} \times \{\mathsf{no}\}) \;\cup\; \mathsf{R} \fatsemi \{(\bot,\bot)\} \;\cup\; \mathsf{R} \fatsemi (\{\omega\} \times \mathsf{G}_\omega) \;\cup\; \{(\bot,\bot)\} \fatsemi \mathsf{CFin}$$
$$\cup\; \{(\bot,\bot)\} \fatsemi (\overline{\mathrm{dom}\,\mathsf{CFin}} \times \{\mathsf{no}\}) \;\cup\; \{(\bot,\bot)\} \fatsemi \{(\bot,\bot)\} \;\cup\; \{(\bot,\bot)\} \fatsemi (\{\omega\} \times \mathsf{G}_\omega)$$
$$\cup\; (\{\omega\} \times \mathsf{CState}_\omega) \fatsemi \mathsf{CFin} \;\cup\; (\{\omega\} \times \mathsf{CState}_\omega) \fatsemi (\overline{\mathrm{dom}\,\mathsf{CFin}} \times \{\mathsf{no}\}) \;\cup\; (\{\omega\} \times \mathsf{CState}_\omega) \fatsemi \{(\bot,\bot)\}$$
$$\cup\; (\{\omega\} \times \mathsf{CState}_\omega) \fatsemi (\{\omega\} \times \mathsf{G}_\omega)$$
$$\subseteq$$
$$\mathsf{AFin} \cup (\overline{\mathrm{dom}\,\mathsf{AFin}} \times \{\mathsf{no}\}) \cup \{(\bot,\bot)\} \cup \{\omega\} \times \mathsf{G}_\omega$$
$$\equiv \{ \;\bot, \omega, \mathsf{no} \notin \text{ domain or range of finalisation or } \mathsf{R} \;\}$$
$$\mathsf{R} \fatsemi \mathsf{CFin} \cup (\mathrm{dom}(\mathsf{R} \mathbin{\rhd\!\!\!-} (\mathrm{dom}\,\mathsf{CFin})) \times \{\mathsf{no}\}) \cup \varnothing \cup \varnothing \cup \varnothing \cup \varnothing \cup \{(\bot,\bot)\} \cup \varnothing \cup (\{\omega\} \times \mathrm{ran}\,\mathsf{CFin})$$
$$\cup\; \{(\omega, \mathsf{no})\} \cup \varnothing \cup \{\omega\} \times \mathsf{G}_\omega$$
$$\subseteq$$
$$\mathsf{AFin} \cup (\overline{\mathrm{dom}\,\mathsf{AFin}} \times \{\mathsf{no}\}) \cup \{(\bot,\bot)\} \cup \{\omega\} \times \mathsf{G}_\omega$$
$$\equiv \{ \text{ calculus } \}$$
$$\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin} \quad\wedge\quad \mathrm{dom}(\mathsf{R} \mathbin{\rhd\!\!\!-} (\mathrm{dom}\,\mathsf{CFin})) \subseteq \overline{\mathrm{dom}\,\mathsf{AFin}}$$
$$\equiv \{ \text{ calculus, taking domains of first conjunct to make second symmetric } \}$$
$$\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin} \quad\wedge\quad (\mathrm{dom}\,\mathsf{AFin}) \lhd \mathsf{R} = \mathsf{R} \rhd (\mathrm{dom}\,\mathsf{CFin})$$

Note that the last step adds some information from the first conjunct onto the second, and thus adds some "finalisation correctness" onto the "finalisation applicability" condition; however, we will still refer to the second condition using the latter name.

**Operations**

We first simplify the compositions of an embedded operation $\mathsf{Op} = (\mathsf{N}, \mathsf{B}, \mathsf{D})$ with an embedded simulation, leading to:

$$\llbracket \mathsf{R} \rrbracket \fatsemi \llbracket \mathsf{Op} \rrbracket = \mathsf{R} \fatsemi \mathsf{N} \;\cup\; \mathsf{R} \fatsemi (\mathsf{B} \times \{\bot\}) \;\cup\; \mathsf{R} \fatsemi (\mathsf{D} \times \mathsf{State}_\omega) \;\cup\; \{(\bot,\bot)\} \;\cup\; \{\omega\} \times \mathsf{State}_{\bot,\omega}$$
$$\llbracket \mathsf{Op} \rrbracket \fatsemi \llbracket \mathsf{R} \rrbracket = \mathsf{N} \fatsemi \mathsf{R} \;\cup\; (\mathsf{B}_{\bot,\omega} \times \{\bot\}) \;\cup\; \mathsf{D}_\omega \times \mathsf{S}_\omega$$

For corresponding operations $\mathsf{AOp} = (\mathsf{AN}, \mathsf{AB}, \mathsf{AD})$ and $\mathsf{COp} = (\mathsf{CN}, \mathsf{CB}, \mathsf{CD})$, we have that

$$\llbracket \mathsf{R} \rrbracket \fatsemi \llbracket \mathsf{COp} \rrbracket \subseteq \llbracket \mathsf{AOp} \rrbracket \fatsemi \llbracket \mathsf{R} \rrbracket$$
$$\equiv \{ \text{ above; simplifications } \}$$
$$\mathsf{R} \fatsemi \mathsf{CN} \;\cup\; \mathsf{R} \fatsemi (\mathsf{CB} \times \{\bot\}) \;\cup\; \mathsf{R} \fatsemi (\mathsf{CD} \times \mathsf{CState}_\omega)$$
$$\subseteq$$
$$\mathsf{AN} \fatsemi \mathsf{R} \;\cup\; (\mathsf{AB} \times \{\bot\}) \;\cup\; \mathsf{AD} \times \mathsf{CState}_\omega$$
$$\equiv \{ \text{ inclusion of set union } \}$$
$$\mathsf{R} \fatsemi \mathsf{CN} \subseteq \mathsf{AN} \fatsemi \mathsf{R} \;\cup\; (\mathsf{AB} \times \{\bot\}) \;\cup\; \mathsf{AD} \times \mathsf{CState}_\omega$$
$$\wedge\; \mathsf{R} \fatsemi (\mathsf{CB} \times \{\bot\}) \subseteq \mathsf{AN} \fatsemi \mathsf{R} \;\cup\; (\mathsf{AB} \times \{\bot\}) \;\cup\; \mathsf{AD} \times \mathsf{CState}_\omega$$
$$\wedge\; \mathsf{R} \fatsemi (\mathsf{CD} \times \mathsf{CState}_\omega) \subseteq \mathsf{AN} \fatsemi \mathsf{R} \;\cup\; (\mathsf{AB} \times \{\bot\}) \;\cup\; \mathsf{AD} \times \mathsf{CState}_\omega$$
$$\equiv \{ \text{ simplification: domains } \}$$
$$\mathsf{R} \fatsemi \mathsf{CN} \subseteq \mathsf{AN} \fatsemi \mathsf{R} \;\cup\; \mathsf{AD} \times \mathsf{CState}$$

$$\wedge\ \mathsf{R} \fatsemi (\mathsf{CB} \times \{\bot\}) \subseteq (\mathsf{AB} \times \{\bot\}) \wedge \mathsf{R} \fatsemi (\mathsf{CD} \times \mathsf{CState}_\omega) \subseteq \mathsf{AD} \times \mathsf{CState}_\omega$$
$$\equiv \{ \text{ relational calculus } \}$$
$$\mathsf{AD} \mathbin{\lhd\!\!\!\!-} \mathsf{R} \fatsemi \mathsf{CN} \subseteq \mathsf{AN} \fatsemi \mathsf{R} \quad \wedge \quad \mathrm{dom}(\mathsf{R} \rhd \mathsf{CB}) \subseteq \mathsf{AB} \quad \wedge \quad \mathrm{dom}(\mathsf{R} \rhd \mathsf{CD}) \subseteq \mathsf{AD}$$

These derivations establish the following.

**Theorem 4 (Downward simulation for process data types).**
The relation $\mathsf{R}$ between $\mathsf{AState}$ and $\mathsf{CState}$ is a downward simulation between the process data types $(\mathsf{AState}, \mathsf{AInits}, \{\mathsf{AOp}_i\}_{i \in I}, \mathsf{AFin})$ and $(\mathsf{CState}, \mathsf{CInits}, \{\mathsf{COp}_i\}_{i \in I}, \mathsf{CFin})$, iff

$$\mathsf{CInits} \subseteq \mathrm{ran}(\mathsf{AInits} \lhd \mathsf{R})$$
$$\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin}$$
$$(\mathrm{dom}\,\mathsf{AFin}) \lhd \mathsf{R} = \mathsf{R} \rhd (\mathrm{dom}\,\mathsf{CFin})$$

and $\forall\, i : I$, for $\mathsf{AOp}_i = (\mathsf{AN}, \mathsf{AB}, \mathsf{AD})$, $\mathsf{COp}_i = (\mathsf{CN}, \mathsf{CB}, \mathsf{CD})$

$$\mathsf{AD} \mathbin{\lhd\!\!\!\!-} \mathsf{R} \fatsemi \mathsf{CN} \subseteq \mathsf{AN} \fatsemi \mathsf{R}$$
$$\mathrm{dom}(\mathsf{R} \rhd \mathsf{CB}) \subseteq \mathsf{AB}$$
$$\mathrm{dom}(\mathsf{R} \rhd \mathsf{CD}) \subseteq \mathsf{AD}$$

□

The rules for initialisation and finalisation are identical to the usual rules[8]. The three rules for operations are the expected generalisations. The first is "correctness", ensuring correct after-states, provided the abstract system does not diverge; in the blocking approach, this proviso is immaterial. The second and third rules both relate to what is normally known as "applicability". When $(\mathsf{B}, \mathsf{D}) = (\varnothing, \overline{\mathrm{dom}\,\mathsf{Op}_i})$ (i.e., the non-blocking interpretation) the second is vacuously true and the third reduces to the usual $\mathrm{ran}(\mathrm{dom}\,\mathsf{AOp}_i \lhd \mathsf{R}) \subseteq \mathrm{dom}\,\mathsf{COp}_i$; in the opposite (blocking) case, the second reduces to the same familiar condition and the third is trivially true.

## 4.2. Upward simulation for process data types

Consider the process data types $(\mathsf{AState}, \mathsf{AInits}, \{\mathsf{AOp}_i\}_{i \in I}, \mathsf{AFin})$ and $(\mathsf{CState}, \mathsf{CInits}, \{\mathsf{COp}_i\}_{i \in I}, \mathsf{CFin})$, both with global state $\mathsf{G}$, and a candidate downward simulation relation $\mathsf{T}$ between $\mathsf{CState}$ and $\mathsf{AState}$, embedded as follows:

$$[\![\mathsf{T}]\!] == \mathsf{T} \cup \{(\bot, \bot)\} \cup \{\omega\} \times \mathsf{AState}_\omega$$

**Initialisation** A simple calculation shows

$$\mathsf{CInit} \fatsemi [\![\mathsf{T}]\!] \subseteq \mathsf{AInit} \ \equiv\ \mathrm{ran}(\mathsf{CInits} \lhd \mathsf{T}) \subseteq \mathsf{AInits}$$

**Finalisation** We derive:

$$[\![\mathsf{CFin}]\!] \subseteq [\![\mathsf{T}]\!] \fatsemi [\![\mathsf{AFin}]\!]$$
$$\equiv \{ \text{ embeddings } \}$$
$$\mathsf{CFin} \cup (\overline{\mathrm{dom}\,\mathsf{CFin}} \times \{\mathsf{no}\}) \cup \{(\bot, \bot)\} \cup \{\omega\} \times \mathsf{G}_\omega$$
$$\subseteq$$
$$(\mathsf{T} \cup \{(\bot, \bot)\} \cup \{\omega\} \times \mathsf{AState}_\omega) \fatsemi (\mathsf{AFin} \cup (\overline{\mathrm{dom}\,\mathsf{AFin}} \times \{\mathsf{no}\}) \cup \{(\bot, \bot)\} \cup \{\omega\} \times \mathsf{G}_\omega$$
$$\equiv \{ \text{ domains; calculus (see also [17, page 80]) } \}$$
$$\mathsf{CFin} \subseteq \mathsf{T} \fatsemi \mathsf{AFin} \ \wedge\ \overline{\mathrm{dom}\,\mathsf{CFin}} \subseteq \mathrm{dom}(\mathsf{T} \mathbin{\rhd\!\!\!\!-} \mathrm{dom}\,\mathsf{AFin})$$

[8] The applicability rule for finalisation is less common, but is also included in [17].

This cannot be further simplified: the first conjunct implies that every finalisable concrete state is linked to *some* finalisable abstract state, and the second similarly links non-finalisable states. However, neither excludes a concrete state being linked to both finalisable and non-finalisable abstract states. Together they do imply totality of $\mathsf{T}$ on $\mathsf{CState}$ as usual.

**Operations**

For corresponding operations $\mathsf{AOp} = (\mathsf{AN}, \mathsf{AB}, \mathsf{AD})$ and $\mathsf{COp} = (\mathsf{CN}, \mathsf{CB}, \mathsf{CD})$, we have that

$$
\begin{array}{l}
[\![\mathsf{COp}]\!] \mathbin{;} [\![\mathsf{T}]\!] \subseteq [\![\mathsf{T}]\!] \mathbin{;} [\![\mathsf{AOp}]\!] \\
\quad \equiv \{ \text{ see downward simulation derivation } \} \\
\mathsf{CN} \mathbin{;} \mathsf{T} \ \cup \ (\mathsf{CB}_\perp \times \{\perp\}) \ \cup \ (\mathsf{CD}_\omega \times \mathsf{AState}_\omega) \cup \{(\omega, \perp)\} \\
\subseteq \\
\mathsf{T} \mathbin{;} \mathsf{AN} \ \cup \ \mathsf{T} \mathbin{;} (\mathsf{AB} \times \{\perp\}) \ \cup \ \mathsf{T} \mathbin{;} (\mathsf{AD} \times \mathsf{AState}_\omega) \ \cup \ \{(\perp, \perp)\} \ \cup \ \{\omega\} \times \mathsf{AState}_{\perp,\omega} \\
\quad \equiv \{ \text{ inclusion of union; domain/range based simplifications } \} \\
\mathsf{CN} \mathbin{;} \mathsf{T} \subseteq \mathsf{T} \mathbin{;} \mathsf{AN} \ \cup \ \mathsf{T} \mathbin{;} (\mathsf{AD} \times \mathsf{AState}) \\
\wedge \ \mathsf{CB} \times \{\perp\} \subseteq \mathsf{T} \mathbin{;} (\mathsf{AB} \times \{\perp\}) \ \wedge \ \mathsf{CD} \times \mathsf{AState}_\omega \subseteq \mathsf{T} \mathbin{;} (\mathsf{AD} \times \mathsf{AState}_\omega) \\
\quad \equiv \{ \text{ calculus } \} \\
\mathrm{dom}(\mathsf{T} \rhd \mathsf{AD}) \mathbin{⩤} \mathsf{CN} \mathbin{;} \mathsf{T} \subseteq \mathsf{T} \mathbin{;} \mathsf{AN} \quad \wedge \quad \mathsf{CB} \subseteq \mathrm{dom}(\mathsf{T} \rhd \mathsf{AB}) \quad \wedge \quad \mathsf{CD} \subseteq \mathrm{dom}(\mathsf{T} \rhd \mathsf{AD})
\end{array}
$$

These derivations establish the following.

**Theorem 5 (Upward simulation for process data types).**
The relation $\mathsf{T}$ between $\mathsf{CState}$ and $\mathsf{AState}$ is an upward simulation between the process data types $(\mathsf{AState}, \mathsf{AInits}, \{\mathsf{AOp}_i\}_{i \in I}, \mathsf{AFin})$ and $(\mathsf{CState}, \mathsf{CInits}, \{\mathsf{COp}_i\}_{i \in I}, \mathsf{CFin})$, iff

$$
\begin{array}{l}
\mathrm{ran}(\mathsf{CInits} \lhd \mathsf{T}) \subseteq \mathsf{AInits} \\
\overline{\mathsf{CFin} \subseteq \mathsf{T} \mathbin{;} \mathsf{AFin}} \\
\mathrm{dom}\,\mathsf{CFin} \subseteq \mathrm{dom}(\mathsf{T} \mathbin{⩥} \mathrm{dom}\,\mathsf{AFin})
\end{array}
$$

and $\forall\, i : I$, for $\mathsf{AOp}_i = (\mathsf{AN}, \mathsf{AB}, \mathsf{AD})$, $\mathsf{COp}_i = (\mathsf{CN}, \mathsf{CB}, \mathsf{CD})$

$$
\begin{array}{l}
\mathrm{dom}(\mathsf{T} \rhd \mathsf{AD}) \mathbin{⩤} \mathsf{CN} \mathbin{;} \mathsf{T} \subseteq \mathsf{T} \mathbin{;} \mathsf{AN} \\
\mathsf{CB} \subseteq \mathrm{dom}(\mathsf{T} \rhd \mathsf{AB}) \\
\mathsf{CD} \subseteq \mathrm{dom}(\mathsf{T} \rhd \mathsf{AD})
\end{array}
$$

□

## 4.3. Simulations on process data types and basic data types

In order to transfer results derived for basic data types, in particular those whose reductions contain an embedding of outputs or refusals, we compare the rules derived above for process data types with the usual definitions of simulations on total data types (Definitions 3 and 4).

As the essential information added in constructing a process data type is attached to the operations only, we would expect the conditions on initialisation and finalisation to be identical, and indeed they are (modulo the differences introduced by process data types using initialisation sets rather than relations, and possible partiality of finalisation). An important consequence of this is in adopting a refusal embedding: this only affects the correctness of finalisation, and so incurs the same extra simulation requirements on process data types as on total ones. Of course this does require the refusal notion to be expressed in terms of triples $(\mathsf{N}, \mathsf{B}, \mathsf{D})$, and the resulting requirement (as in [18]) may still end up dominating some of the other conditions on operations. The property (see [17]) that finalisation conditions resulting from an output embedding are trivially true also transfers across.

## 5. Failures-divergences with internal operations

In the previous section, we derived simulation rules for the generalised process data type by embedding it into a relational model with explicit error values $\perp$ and $\omega$. This section presents the major result of this paper: simulations for relational failures-divergences refinement of data types extended with an internal operation, defined through an embedding in process data types. There are two versions of this: the blocking and the non-blocking approaches, both deriving from the same process data type simulation rules. This reconstructs the approach in [19] which inspired the definition of process data types. The following section will then revisit these two approaches, adding outputs to the data types.

### 5.1. Basic data type with internal operation

We develop a relational verification method for failures-divergences refinement. Thus, we assume the basic data type is a refusal embedding – in the first instance in the blocking approach for the basic refusal relation characterised by

$$\forall\, s : \mathsf{State}, E : \mathbb{P}\, I \bullet\ (s, E) \in \mathsf{Ref}_{\mathsf{State}} \ \equiv\ (\forall\, i : E \bullet s \notin \operatorname{dom} \mathsf{Op}_i)$$

**Definition 11 (Basic data type with internal operation).**
A basic data type with internal operation is a quintuple $(\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i \in I}, \tau, \mathsf{Fin})$ such that $(\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i \in I}, \mathsf{Fin})$ is a basic data type, and the internal operation $\tau$ is a relation on $\mathsf{State}$. □

### 5.2. Embedding into process data types

The addition of internal operations introduces a large number of, sometimes subtle, issues (see also [20, 17]):

a) a finite number of internal operations may take place before and after every operation;
b) in some states, unbounded internal evolution may be enabled (livelock), making those states and the operations leading into those states divergent;
c) initial states may also be divergent; if any one initial state is, all traces of the ADT are divergent;
d) the CSP failures-divergences semantics only observes refusals in stable states, i.e., where no internal evolution is possible. However, from the fact that traces involving unstable states are observable, enabledness of operations in such states may be observable; blocking in unstable states is more subtle, however. It also means that observing refusals at finalisation needs to be treated with care, in case the final state is not stable;
e) although refusal of operations in unstable states is immaterial, the opposite, i.e., enabled operations in unstable states is *not*: we need to ensure that traces arising from operations being "momentarily" available in unstable states are included.

**Notation** We define the following auxiliary notations in order to deal with internal evolution. This includes notations for (maximal) finite internal evolution which are used to enhance the operations (etc).

$\mathsf{State}{\downarrow}$ denotes the set of stable states, i.e. those from which it is not possible to do an internal evolution. That is, $v \in \mathsf{State}{\downarrow}$ iff $v \notin \operatorname{dom} \tau$.

$\tau^*$ denotes finite internal evolution, defined as the least fixed point of $\lambda\, \mathsf{R} \bullet \mathsf{id}_{\mathsf{State}} \cup \tau \mathbin{⨾} \mathsf{R}$.

$\tau^{*|} == \tau^* \mathbin{\rhd} (\operatorname{dom} \tau)$ denotes maximal finite internal evolution, leading to a stable state. Note that all unstable states either are divergent, or are linked by finite internal evolution to a non-empty set of stable states.

$\overleftrightarrow{\mathsf{Op}} == \tau^* \fatsemi \mathsf{Op} \fatsemi \tau^*$ represents an operation with internal evolution beforehand and afterwards. Note that $\mathrm{dom}\,\overleftrightarrow{\mathsf{Op}} = \mathrm{dom}(\tau^* \fatsemi \mathsf{Op})$ as $\mathrm{dom}\,\tau^* = \mathsf{State}$.

$\mathsf{State}\!\uparrow$ denotes the set of "divergent" states from which unbounded internal evolution is possible. It is defined as the largest fixed point of $\lambda\,\mathsf{S}.\,\mathrm{dom}(\tau \rhd \mathsf{S})$. If any initial states are in $\mathsf{State}\!\uparrow$, the ADT as a whole is divergent.

$\underline{\tau} == (\mathsf{State}\!\uparrow) \ntriangleleft \tau$ denotes "relevant" internal transitions, excluding those from states which are already divergent. As everything is possible after divergence, the presence of finite internal behaviour from such states is semantically insignificant. Note that $\underline{\tau}^* = \mathsf{id}_{\mathsf{State}\uparrow} \cup (\mathsf{State}\!\uparrow) \ntriangleleft \tau^*$.

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

$$\mathrm{liv}\,\mathsf{Op} = \mathrm{dom}(\mathsf{Op} \rhd \mathsf{State}\!\uparrow)$$

Note that these are not necessarily states which are *themselves* divergent, but states from where $\mathsf{Op}$ might lead to divergent states.

Using these notations, we can construct embeddings of basic data types with an internal operation into process data types for the blocking and non-blocking approaches. This gives a semantics for these types, and we will later validate that, in combination with the right refusal embedding, this correctly encodes failures-divergences semantics.

**Definition 12 (Embeddings of internal operations into process data type).**
A basic data type with internal operation $(\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i\in I}, \tau, \mathsf{Fin})$ is embedded into the process data type $(\mathsf{State}, \widetilde{\mathsf{Init}}, \{\widetilde{\mathsf{Op}_i}\}_{i\in I}, \widetilde{\mathsf{Fin}})$ with

$$\widetilde{\mathsf{Init}} == \mathrm{ran}(\mathsf{Init} \fatsemi \tau^*)$$
$$\widetilde{\mathsf{Fin}} == \mathsf{State}\!\uparrow \ntriangleleft \tau^{*|} \fatsemi \mathsf{Fin}$$
$$\widetilde{\mathsf{Op}_i} == (\mathrm{liv}\,\overleftrightarrow{\mathsf{Op}}_i \ntriangleleft \overleftrightarrow{\mathsf{Op}}_i, \mathsf{B}_i, \mathsf{D}_i)$$

where in the blocking approach

$$\mathsf{B}_i == \overline{\mathrm{dom}\,\overleftrightarrow{\mathsf{Op}}_i} \qquad \mathsf{D}_i == \mathrm{liv}\,\overleftrightarrow{\mathsf{Op}}_i$$

and in the non-blocking approach

$$\mathsf{B}_i == \varnothing \qquad \mathsf{D}_i == \mathrm{liv}\,\overleftrightarrow{\mathsf{Op}}_i \cup \overline{\mathrm{dom}\,\overleftrightarrow{\mathsf{Op}}_i}$$

□

We argue informally that this embedding addresses each of the issues listed on page 23, with the exception of issue c) concerning possible divergence of initial states:

a) Using $\overleftrightarrow{\mathsf{Op}}$ ensures finite internal evolution before and after every operation.
b) By including $\mathrm{liv}\,\overleftrightarrow{\mathsf{Op}}$ in the divergence domain of an operation, we ensure that livelock becomes divergence.
c) This encoding does *not* deal with divergent initial states. The process data type could be enhanced with a Boolean to represent initial divergence (and include $\omega$ in $\mathsf{Inits}$ whenever it is set), however, this pathological situation can more easily be singled out in the definition of refinement.
d) In our previous work [20, 17] it was sufficient to include internal operations after each operation and initialisation only – the need to include internal operations *before* derives from observing refusals and blocking (in the blocking approach) and divergence outside the precondition (in the non-blocking approach) in stable states only.[9] In general, in the context of *stable* failures, treating an operation that

[9] The corresponding encoding in [19] is erroneous in this respect, and causes the relational semantics to make incorrect distinctions by including unstable refusals. This became evident in mechanising the proof.

is not enabled as one that has an artificial result ($\perp$) does not work across the board – it depends on whether $\tau$ is enabled in the same state or not.
The finalisation, additionally, is preceded by maximal finite internal evolution in order to avoid observing refusals in unstable states, and restricted to non-divergent states. Alternatively, we could restrict finalisation to stable states only[10].
Note that a finalisation which observes refusals in the non-blocking case is quite different from the blocking case, as there are no refusals in the non-blocking approach unless outputs are taken into account.

e) This is addressed by including finite, but *not* necessarily maximal internal behaviour after every operation (and initialisation) – i.e., not considering only the stable after-states.

## 5.3. Refinement

The definition of refinement for basic data types with internal operations is given in terms of the embedding into process data types, with a small proviso to deal with divergent initial states.

**Definition 13 (Refinement with internal operations).**
A program controlled basic data type with internal operation $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in I}, \tau_{\mathsf{A}}, \mathsf{AFin})$ is refined by another such data type $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in I}, \tau_{\mathsf{C}}, \mathsf{CFin})$ iff

- One of A's initial states is divergent: $(\operatorname{ran} \mathsf{AInit}) \cap \mathsf{AState}\!\uparrow\ \neq \varnothing$, or
- None of C's initial states are divergent, i.e., $(\operatorname{ran} \mathsf{CInit}) \cap \mathsf{CState}\!\uparrow\ = \varnothing$, and:
  refinement holds between the embeddings of A and C in their embeddings into process data types. $\square$

By separating out divergent initial states in this way, we can restrict our discussion of simulations to those cases where the process data type embedding correctly reflects the original specification in all respects.

## 5.4. Correctness of the embedding: blocking approach

First, however, we prove that the embedding of internal operations into process data types, and through that into total data types, correctly reflects the failures-divergences semantics in the blocking approach (which is the more common one for a concurrency context).

In order to do so, we first characterise formally the failures-divergences semantics of a data type with internal operations (ignoring its finalisation). We use some standard labeled transition notation to facilitate that, as follows. Note that (only) the definition of $\xrightarrow{e}$ will become more complicated later when we also use outputs.

**Definition 14 (Transition notation).**
For a data type with internal operations $(\mathsf{State}, \mathsf{G} \times \mathsf{Inits}, \{\mathsf{Op}_i\}_{i \in I}, \tau, \mathsf{Fin})$ we define the following notations ($\varepsilon$ denotes the empty trace):

$$
\begin{array}{llll}
\mathsf{x} \xrightarrow{i} \mathsf{x}' \ == \ (\mathsf{x}, \mathsf{x}') \in \mathsf{Op}_i & \qquad & \mathsf{x} \xrightarrow{i} \ == \ \exists\, \mathsf{x}' \bullet \mathsf{x} \xrightarrow{i} \mathsf{x}' \\
\mathsf{x} \xrightarrow{\tau} \mathsf{x}' \ == \ (\mathsf{x}, \mathsf{x}') \in \tau & \qquad & \mathsf{x} \not\xrightarrow{i} \ == \ \neg \mathsf{x} \xrightarrow{i}
\end{array}
$$

and $\Longrightarrow$ is the transitive reflexive closure of $\longrightarrow$, collecting all the labels in a sequence except for $\tau$:

$\mathsf{x} \overset{s}{\Longrightarrow} \mathsf{x}' \ == \ \exists\, p : \operatorname{seq}(I \cup \{\tau\}) \bullet \mathsf{x} \overset{p}{\twoheadrightarrow} \mathsf{x}' \wedge s = p \upharpoonright I$
where

[10] This works because $\tau^{*|} \fatsemi \mathsf{Fin} = \tau^* \rhd (\operatorname{dom} \tau) \fatsemi \mathsf{Fin} = \tau^* \fatsemi (\operatorname{dom} \tau) \lhd \mathsf{Fin}$, and $\tau^*$ is always absorbed by preceding $\tau^*$.

$$\mathsf{x} \overset{\langle\rangle}{\twoheadrightarrow} \mathsf{x}' \;==\; \mathsf{x} = \mathsf{x}' \qquad\qquad \mathsf{x} \overset{p \frown q}{\twoheadrightarrow} \mathsf{x}' \;==\; \exists\, \mathsf{x}'' \bullet \mathsf{x} \overset{p}{\twoheadrightarrow} \mathsf{x}'' \land \mathsf{x}'' \overset{q}{\twoheadrightarrow} \mathsf{x}'$$

and an omitted first state indicates an initial one:

$$\overset{s}{\Longrightarrow} \mathsf{x} \;==\; \exists\, \mathsf{x}' : \mathsf{Inits} \bullet \mathsf{x}' \overset{s}{\Longrightarrow} \mathsf{x} \qquad \Box$$

Failures and divergences are then defined in the usual way. Note that they would be different in the non-blocking approach, producing no refusals but divergences instead.

**Definition 15 (Failures-divergences semantics).**
For a data type with internal operations $\mathsf{T} = (\mathsf{State}, \mathsf{G} \times \mathsf{Inits}, \{\mathsf{Op}_i\}_{i \in I}, \tau, \mathsf{Fin})$ its divergences are

$$\mathit{div}(\mathsf{T}) == \{s : \operatorname{seq} I \mid \exists\, s' : \operatorname{seq} I;\ \mathsf{x} : \mathsf{State}\!\uparrow\ \bullet s' \leq s \land \overset{s'}{\Longrightarrow} \mathsf{x}\}$$

and its failures are

$$f(\mathsf{T}) == \{s : \operatorname{seq} I;\ E : \mathbb{P}\, I \mid s \in \mathit{div}(\mathsf{T}) \lor (\exists\, \mathsf{x} : \mathsf{State}\!\downarrow\ \bullet \overset{s}{\Longrightarrow} \mathsf{x} \land \forall\, e : E \bullet \mathsf{x} \overset{e}{\not\longrightarrow}) \bullet (s, E)\} \qquad \Box$$

The correctness of the embedding will be proved using the following lemmas. Recall that we are working in a context where:

- $\mathsf{T}$ is defined as in Definition 15; and is program controlled, so $\mathsf{Init} \equiv \mathsf{G} \times \operatorname{ran} \mathsf{Init}$;
- we use the blocking interpretation of $\mathsf{T}$;
- $\mathsf{T}$ observes refusals at finalisation;
- $\mathsf{T}$ is not initially divergent.

We have not defined failures and divergences at the level of process data types, and therefore we will have to prove the equivalence of refinement accounting for *two* levels of embedding. Recall that $\widetilde{x}$ is the embedding of $x$ into process data types (Definition 12), and $[\![x]\!]$ is the embedding of $x$ from process data types into basic data types (Definition 10)[11].

**Lemma 1.** For all programs $p : \operatorname{seq} I$, we have that

$$p \in \mathit{div}(\mathsf{T}) \equiv \omega \in \operatorname{ran} p_{[\![\widetilde{\mathsf{T}}]\!]}$$

**Proof** A simple proof shows that

$$\omega \in \operatorname{ran} p_{[\![\widetilde{\mathsf{T}}]\!]} \;\equiv\; \omega \in \operatorname{ran}([\![\widetilde{\mathsf{Init}}]\!] \mathbin{;} [\![\widetilde{p}]\!])$$

Because $\mathsf{T}$ is not initially divergent, $p$ is non-empty. We now prove

$$\omega \in \operatorname{ran}([\![\widetilde{\mathsf{Init}}]\!] \mathbin{;} [\![\widetilde{p}]\!]) \equiv p \in \mathit{div}(\mathsf{T})$$

by induction over the construction of $p$.

$$\begin{aligned}
&\omega \in \operatorname{ran}([\![\widetilde{\mathsf{Init}}]\!] \mathbin{;} [\![\widetilde{p}]\!]) \\
&\qquad \equiv \{\text{ Let } p = p' \frown \langle i \rangle \ \} \\
&\omega \in \operatorname{ran}([\![\widetilde{\mathsf{Init}}]\!] \mathbin{;} [\![\widetilde{p'}]\!] \mathbin{;} [\![\widetilde{\mathsf{Op}_i}]\!]) \\
&\qquad \equiv \{\text{ definition of } \widetilde{\mathsf{Op}_i} \ \} \\
&\omega \in \operatorname{ran}([\![\widetilde{\mathsf{Init}}]\!] \mathbin{;} [\![\widetilde{p'}]\!] \mathbin{;} [\![(\operatorname{liv} \overleftrightarrow{\mathsf{Op}}_i \lhd \overleftrightarrow{\mathsf{Op}}_i, \overline{\operatorname{dom} \overleftrightarrow{\mathsf{Op}}_i}, \operatorname{liv} \overleftrightarrow{\mathsf{Op}}_i)]\!]
\end{aligned}$$

[11] This is different from the precursor paper [19] where the various embeddings directly into basic data types were defined as $\widehat{x}^t$ with varying $t$, and $\tilde{x}$ was used for embedding simulations.

$$\equiv \{ \text{ definition of } [\![(\mathsf{N}, \mathsf{B}, \mathsf{D})]\!]\text{: } \omega \text{ only from } \mathsf{D} \text{ and } \omega \ \}$$

$$\omega \in \mathrm{ran}([\![\widetilde{\mathsf{Init}}]\!] \,\S\, [\![\widetilde{p'}]\!]) \vee \mathrm{ran}([\![\widetilde{\mathsf{Init}}]\!] \,\S\, [\![\widetilde{p'}]\!]) \cap \mathrm{liv}\, \overleftrightarrow{\mathsf{Op}}_i \neq \varnothing$$

The first disjunct can only hold when $p'$ is nonempty (because $\mathsf{T}$ is not initially divergent). In that case, by induction $p' \in \mathit{div}(\mathsf{T})$ and, as divergences are closed under extension, $p \in \mathit{div}(\mathsf{T})$ as required. For the second disjunct (including the base case), we have:

$$\mathrm{ran}([\![\widetilde{\mathsf{Init}}]\!] \,\S\, [\![\widetilde{p'}]\!]) \cap \mathrm{liv}\, \overleftrightarrow{\mathsf{Op}}_i \neq \varnothing$$

$$\equiv \{ \text{ definition of } \overleftrightarrow{\mathsf{Op}}_i \text{ and liv; relations } \}$$

$$[\![\widetilde{\mathsf{Init}}]\!] \,\S\, [\![\widetilde{p'}]\!] \,\S\, (\tau^* \,\S\, \mathsf{Op}_i \,\S\, \tau^*) \rhd \mathsf{State}\!\uparrow\ \neq \varnothing$$

$$\equiv \{ \ [\![\widetilde{p'}]\!] \text{ (in the base case: } [\![\widetilde{\mathsf{Init}}]\!]) \text{ and } \mathsf{State}\!\uparrow \text{ closed under composition with } \tau^* \}$$

$$[\![\widetilde{\mathsf{Init}}]\!] \,\S\, [\![\widetilde{p'}]\!] \,\S\, \mathsf{Op}_i \rhd \mathsf{State}\!\uparrow\ \neq \varnothing$$

$$\equiv \{ \text{ definition of } \Longrightarrow\text{; no blocking in } p' \text{ as } \perp \notin \mathrm{dom}\, \mathsf{Op}_i \}$$

$$\exists\, \mathsf{x} : \mathsf{State}\!\uparrow\ \bullet\ \stackrel{p}{\Longrightarrow} \mathsf{x}$$

$$\Rightarrow \{ \text{ definition of } \mathit{div} \ \}$$

$$p \in \mathit{div}(\mathsf{T})$$

We have now proved that $p = p' \frown \langle i \rangle$ is divergent when $p'$ is, and that $p$ is divergent when $p'$ is not and $\mathsf{Op}_i$ diverges in some final state of $p'$. As these are the only two cases in which $p$ could be divergent, we have equivalence (rather than just implication) as required. □

**Lemma 2.** For all programs $p : \mathrm{seq}\, I$ and sets of events $E : \mathbb{P}\, I$ such that $p \notin \mathit{div}(\mathsf{T})$, we have that

$$(p, E) \in f(\mathsf{T}) \equiv E \in \mathrm{ran}\, p_{[\![\widetilde{\mathsf{T}}]\!]}$$

**Proof** We generalise the definition of (non-divergent) refusals to

$$f'(\mathsf{S} : \mathbb{P}\,\mathsf{State}) == \{ s : \mathrm{seq}\, I;\ E : \mathbb{P}\, I \mid \exists\, \mathsf{x} : \mathsf{State}\!\downarrow;\ \mathsf{y} : \mathsf{S} \bullet \mathsf{y} \stackrel{s}{\Longrightarrow} \mathsf{x} \wedge \forall\, e : E \bullet \mathsf{x} \stackrel{e}{\not\longrightarrow} ) \}$$

and the lemma to

$$\omega \notin \mathit{ran}(\mathsf{S} \lhd [\![\widetilde{p}]\!]) \wedge \mathsf{S} \cap \mathsf{State}\!\uparrow\ = \varnothing \quad \Rightarrow \quad (p, E) \in f'(\mathsf{S}) \equiv E \in \mathrm{ran}(\mathsf{S} \lhd [\![\widetilde{p}]\!] \,\S\, [\![\widetilde{\mathsf{Fin}}]\!])$$

which proves the lemma when we instantiate $\mathsf{S}$ to $\mathrm{ran}\, \mathsf{Init}$. The generalisation is proved by induction on $p$.

**Base case** When $p = \langle\, \rangle$ we have:

$$(p, E) \in f'(\mathsf{S})$$

$$\equiv \{ \text{ base case; definition of } f' \ \}$$

$$\exists\, \mathsf{x} : \mathsf{State}\!\downarrow;\ \mathsf{y} : \mathsf{S} \bullet \mathsf{y} \stackrel{\varepsilon}{\Longrightarrow} \mathsf{x} \wedge \forall\, e : E \bullet \mathsf{x} \stackrel{e}{\not\longrightarrow}$$

$$\equiv \{ \text{ definition of } \mathsf{Fin}\text{, definition of } \tau^{*|} \ \}$$

$$E \in \mathrm{ran}(\mathsf{S} \lhd (\mathsf{State}\!\uparrow\ \ntriangleleft \tau^{*|} \,\S\, \mathsf{Fin}))$$

$$\equiv \{ \text{ embeddings of } \mathsf{Fin}\text{; insert empty program } \}$$

$$E \in \mathrm{ran}(\mathsf{S} \lhd [\![\widetilde{p}]\!] \,\S\, [\![\widetilde{\mathsf{Fin}}]\!])$$

**Induction step** Let $p = \langle i \rangle \frown p'$.

$$(p, E) \in f'(\mathsf{S})$$

$$\equiv \{ \text{ definition of } f' \ \}$$

$$\exists\, \mathsf{x} : \mathsf{State}\!\downarrow;\ \mathsf{y} : \mathsf{S} \bullet \mathsf{y} \stackrel{\langle i \rangle \frown p'}{\Longrightarrow} \mathsf{x} \wedge \forall\, e : E \bullet \mathsf{x} \stackrel{e}{\not\longrightarrow}$$

$$\equiv \{ \text{ definition of } \Longrightarrow \text{; relational calculus } \}$$

$$\exists \mathsf{x} : \mathsf{State}\!\downarrow\, ; \ \mathsf{y}' : \mathrm{ran}(\mathsf{S} \lhd \overleftrightarrow{\mathsf{Op}}_i) \bullet \mathsf{y}' \overset{p'}{\Longrightarrow} \mathsf{x} \land \forall\, e : E \bullet \mathsf{x} \overset{e}{\not\longrightarrow}$$
$$\equiv \{ \text{ definition of } f' \ \}$$
$$(p', E) \in f'(\mathrm{ran}(\mathsf{S} \lhd \overleftrightarrow{\mathsf{Op}}_i))$$
$$\equiv \{ \text{ induction } \}$$
$$E \in \mathrm{ran}(\mathrm{ran}(\mathsf{S} \lhd \overleftrightarrow{\mathsf{Op}}_i) \lhd [\![\widetilde{p'}]\!] \mathbin{;} [\![\widetilde{\mathsf{Fin}}]\!])$$
$$\equiv \{ \text{ embeddings of operations } \}$$
$$E \in \mathrm{ran}(\mathsf{S} \lhd [\![\widetilde{\mathsf{Op}_i}]\!] \mathbin{;} [\![\widetilde{p'}]\!] \mathbin{;} [\![\widetilde{\mathsf{Fin}}]\!])$$
$$\equiv \{ \text{ program composition } \}$$
$$E \in \mathrm{ran}(\mathsf{S} \lhd [\![\widetilde{p}]\!] \mathbin{;} [\![\widetilde{\mathsf{Fin}}]\!])$$

This completes the proof of Lemma 2. □

Note that, on first impression maybe surprisingly, the two lemmas capture all the relevant information (failures and divergences) without making any reference to $\perp$ in the relational semantics. In fact, the information provided by $\perp$ is redundant when refusals are observed at finalisation. Indeed, in [18] the simulation conditions derived from finalisations which observe refusals dominate the "applicability" conditions (derived by eliminating $\perp$), pointing at a similar redundancy in the situation without internal operations. The redundancy here is formalised in the following lemma (a healthiness condition).

**Lemma 3.** When no strict prefix $p'$ of $p$ is in $div(\mathsf{T})$, then

$$\perp \in \mathrm{ran}\, p_{[\![\widetilde{\mathsf{T}}]\!]} \ \equiv\ \exists\, p', p'', i \bullet p = p' \frown \langle i \rangle \frown p'' \land \{i\} \in \mathrm{ran}\, p'_{[\![\widetilde{\mathsf{T}}]\!]}$$

i.e., when $p$ is not a trace that already diverged previously and it may lead to blocking, then somewhere in $p$ there has to be an index $i$ which could be refused at that point. □

**Proof** Similar to Lemma 2, generalising $\mathrm{ran}\,\mathsf{Init}$ in $p_{[\![\widetilde{\mathsf{T}}]\!]}$ to an arbitrary set and then by induction over $p$. □

The main correctness theorem for the embedding is the following.

**Theorem 6 (Equivalence of failures-divergences and data refinement).**
For two data types $\mathsf{A}$ and $\mathsf{C}$, failures-divergences refinement holds iff data refinement holds between their embeddings.

**Proof** By mutual implication. The proof that relational refinement implies failures-divergences refinement uses Lemma 1 for divergences, and Lemma 2 for failures. The reverse direction has as its demonstrandum

$$x \in \mathrm{ran}\, p_{[\![\widetilde{\mathsf{C}}]\!]} \Rightarrow x \in \mathrm{ran}\, p_{[\![\widetilde{\mathsf{A}}]\!]}$$

which is proved by case distinction on $x$ ($\perp$, $\omega$, $\mathsf{no}$ or a refusal $E$), assuming failures-divergences refinement and using all three lemmas. □

### 5.5. Simulations in the blocking approach

As failures-divergences refinement is correctly represented by relational refinement, we can instantiate simulation rules from Theorems 4 and 5 into simulation rules for data types with internal operation.

### *5.5.1. Downward simulation*

Instantiating Theorem 4 with the embedding of Definition 12 initially gives the following conditions.

$$\mathsf{CInit} \fatsemi \tau_{\mathsf{C}}^{*} \subseteq \mathsf{AInit} \fatsemi \tau_{\mathsf{A}}^{*} \fatsemi \mathsf{R} \tag{3}$$

$$\mathsf{R} \fatsemi \mathsf{CState}\!\uparrow \ndres \tau_{\mathsf{C}}^{*|} \fatsemi \mathsf{CFin} \subseteq \mathsf{AState}\!\uparrow \ndres \tau_{\mathsf{A}}^{*|} \fatsemi \mathsf{AFin} \tag{4}$$

$$(\mathsf{AState}\!\uparrow \ndres \mathsf{R}) = (\mathsf{R} \nrres \mathsf{CState}\!\uparrow) \tag{5}$$

and for matching operations $\mathsf{AOp}$ and $\mathsf{COp}$:

$$(\mathrm{liv}\,\overleftrightarrow{\mathsf{AOp}} \ndres \mathsf{R} \nrres \mathrm{liv}\,\overleftrightarrow{\mathsf{COp}}) \fatsemi \overleftrightarrow{\mathsf{COp}} \subseteq \overleftrightarrow{\mathsf{AOp}} \fatsemi \mathsf{R} \tag{6}$$

$$\mathrm{dom}(\mathsf{R} \rres \mathrm{liv}\,\overleftrightarrow{\mathsf{COp}}) \subseteq \mathrm{liv}\,\overleftrightarrow{\mathsf{AOp}} \tag{7}$$

$$\mathrm{ran}(\mathrm{dom}\,\overleftrightarrow{\mathsf{AOp}} \dres \mathsf{R}) \subseteq \mathrm{dom}\,\overleftrightarrow{\mathsf{COp}} \tag{8}$$

In order to simplify these, in particular to take into account the particular nature of finalisation (observing refusals), we use the following theorem.

**Theorem 7 (Downward simulation closed under $\underline{\tau}_{\mathsf{C}}^{*}$).**
If the relation $\mathsf{R} \subseteq \mathsf{AState} \times \mathsf{CState}$ is a downward simulation between program controlled basic data types with internal operations $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in I}, \tau_{\mathsf{A}}, \mathsf{AFin})$ and $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in I}, \tau_{\mathsf{C}}, \mathsf{CFin})$ then $\mathsf{R} \fatsemi \underline{\tau}_{\mathsf{C}}^{*}$ is also a downward simulation between $\mathsf{A}$ and $\mathsf{C}$.

**Proof** By showing that (3) to (8) above imply the same conditions with $\mathsf{R} \fatsemi \underline{\tau}_{\mathsf{C}}^{*}$ substituted for $\mathsf{R}$. □

The relevance of the theorem is that we can, without loss of generality, restrict ourselves to retrieve relations $\mathsf{R}$ which satisfy $\mathsf{R} = \mathsf{R} \fatsemi \underline{\tau}_{\mathsf{C}}^{*}$. The initialisation condition (3) can then be simplified to $\mathsf{CInit} \subseteq \mathsf{AInit} \fatsemi \tau_{\mathsf{A}}^{*} \fatsemi \mathsf{R}$. The "correctness" condition (6) can be simplified using the divergence condition (7) to

$$(\mathrm{liv}\,\overleftrightarrow{\mathsf{AOp}} \ndres \mathsf{R}) \fatsemi \overleftrightarrow{\mathsf{COp}} \subseteq \overleftrightarrow{\mathsf{AOp}} \fatsemi \mathsf{R}$$

The "finalisation" condition (4) is implied by the "blocking" condition (8) and finalisation applicability (5). Because the refusal finalisation refers explicitly to sets of refused operations, this proof is performed at the predicate calculus level:

$$\begin{array}{l}
\mathsf{R} \fatsemi \mathsf{CState}\!\uparrow \ndres \tau_{\mathsf{C}}^{*|} \fatsemi \mathsf{CFin} \subseteq \mathsf{AState}\!\uparrow \ndres \tau_{\mathsf{A}}^{*|} \fatsemi \mathsf{AFin} \\
\qquad \equiv \{ \text{ definition of } \subseteq \} \\
\forall\, \mathsf{a} : \mathsf{AState};\ \mathsf{E} : \mathbb{P}\, I \bullet (\exists\, \mathsf{c} : \mathsf{CState} \bullet (\mathsf{a}, \mathsf{c}) \in \mathsf{R} \wedge \mathsf{c} \notin \mathsf{CState}\!\uparrow \wedge \exists\, \mathsf{c}' : \mathsf{CState} \bullet (\mathsf{c}, \mathsf{c}') \in \tau_{\mathsf{C}}^{*|} \wedge (\mathsf{c}', \mathsf{E}) \in \mathsf{Ref}) \\
\qquad\qquad\qquad\qquad\qquad \Rightarrow \mathsf{a} \notin \mathsf{AState}\!\uparrow \wedge \exists\, \mathsf{a}' : \mathsf{AState} \bullet (\mathsf{a}, \mathsf{a}') \in \tau_{\mathsf{A}}^{*|} \wedge (\mathsf{a}', \mathsf{E}) \in \mathsf{Ref} \\
\qquad \equiv \{ \text{ predicate calculus } \} \\
\forall\, \mathsf{a} : \mathsf{AState};\ \mathsf{E} : \mathbb{P}\, I;\ \mathsf{c}, \mathsf{c}' : \mathsf{CState} \bullet (\mathsf{a}, \mathsf{c}) \in \mathsf{R} \wedge \mathsf{c} \notin \mathsf{CState}\!\uparrow \wedge (\mathsf{c}, \mathsf{c}') \in \tau_{\mathsf{C}}^{*|} \wedge (\mathsf{c}', \mathsf{E}) \in \mathsf{Ref} \\
\qquad\qquad\qquad\qquad\qquad \Rightarrow \mathsf{a} \notin \mathsf{AState}\!\uparrow \wedge \exists\, \mathsf{a}' : \mathsf{AState} \bullet (\mathsf{a}, \mathsf{a}') \in \tau_{\mathsf{A}}^{*|} \wedge (\mathsf{a}', \mathsf{E}) \in \mathsf{Ref}
\end{array}$$

We prove this by contradiction. Assume the implicand holds, but not the consequent. Thus, $(\mathsf{a}', \mathsf{E}) \notin \mathsf{Ref}_{\mathsf{A}}$ so for some $i \in \mathsf{E}$ we have that $\mathsf{a}' \in \mathrm{dom}\,\mathsf{AOp}_i$. Theorem 7 implies that $(\mathsf{a}, \mathsf{c}') \in \mathsf{R}$; from the blocking condition (8) it then follows that $\mathsf{c}' \in \mathrm{dom}\,\overleftrightarrow{\mathsf{COp}}_i$, and because $c'$ is stable, also $\mathsf{c}' \in \mathrm{dom}\,\mathsf{COp}_i$, contradicting $(\mathsf{c}', \mathsf{E}) \in \mathsf{Ref}_{\mathsf{C}}$. This completes the proof that the finalisation correctness condition is dominated by the other conditions. The simplifications are summarised in the following theorem.

**Theorem 8 (Downward simulation for data types with internal operations (blocking)).** The relation $\mathsf{R}' \subseteq \mathsf{AState} \times \mathsf{CState}$ is a downward simulation between program controlled basic data types with internal operations $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in I}, \tau_\mathsf{A}, \mathsf{AFin})$ and $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in I}, \tau_\mathsf{C}, \mathsf{CFin})$ iff the following conditions hold for $\mathsf{R} == \mathsf{R}' \mathbin{⨾} \underline{\tau}^*_\mathsf{C}$:

$$\mathsf{CInit} \subseteq \mathsf{AInit} \mathbin{⨾} \tau^*_\mathsf{A} \mathbin{⨾} \mathsf{R} \tag{9}$$

$$(\mathsf{AState}\uparrow \lhd \mathsf{R}) = (\mathsf{R} \rhd \mathsf{CState}\uparrow) \tag{10}$$

and for matching operations $\mathsf{AOp}$ and $\mathsf{COp}$:

$$(\mathrm{liv}\, \overleftrightarrow{\mathsf{AOp}} \mathbin{\lhd\!\!\!-} \mathsf{R}) \mathbin{⨾} \overleftrightarrow{\mathsf{COp}} \subseteq \overleftrightarrow{\mathsf{AOp}} \mathbin{⨾} \mathsf{R} \tag{11}$$

$$\mathrm{dom}(\mathsf{R} \rhd \mathrm{liv}\, \overleftrightarrow{\mathsf{COp}}) \subseteq \mathrm{liv}\, \overleftrightarrow{\mathsf{AOp}} \tag{12}$$

$$\mathrm{ran}(\mathrm{dom}\, \overleftrightarrow{\mathsf{AOp}} \lhd \mathsf{R}) \subseteq \mathrm{dom}\, \overleftrightarrow{\mathsf{COp}} \tag{13}$$

□

For specifications in Z, embedded into relations in the usual way, this leads to the following conditions:

**DS.InitBlock.internal** $\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \mathbin{⨾} \tau^*_A \wedge R'$

**DS.AppBlock.internal** $\forall\, CState;\ AState;\ i : I \bullet R \wedge \mathrm{pre}\, \overleftrightarrow{AOp_i} \Rightarrow \mathrm{pre}\, \overleftrightarrow{COp_i}$

**DS.CorrBlock.internal**

$\forall\, i : I;\ AState;\ CState;\ CState' \bullet \neg\, \mathrm{liv}\, \overleftrightarrow{AOp_i} \wedge R \wedge \overleftrightarrow{COp_i} \Rightarrow \exists\, AState' \bullet R' \wedge \overleftrightarrow{AOp_i}$

**DS.DivStates** $\forall\, AState;\ CState \mid R \bullet AState\uparrow \Leftrightarrow CState\uparrow$

**DS.DivOp** $\forall\, CState;\ AState;\ i : I \bullet R \wedge \mathrm{liv}\, \overleftrightarrow{COp_i} \Rightarrow \mathrm{liv}\, \overleftrightarrow{AOp_i}$

using the obvious Z versions of notions such as $\overleftrightarrow{Op}$ and liv $Op$, and assuming $R = R \mathbin{⨾} \underline{\tau}^*_C$. Note, that in the absence of internal operations the first three conditions collapse to the corresponding conditions given in Section 3.3 above.

### *5.5.2. Upward simulation*

Instantiating Theorem 5 with the embedding of Definition 12 initially gives the following conditions.

$$\mathsf{CInit} \mathbin{⨾} \tau^*_\mathsf{C} \mathbin{⨾} \mathsf{T} \subseteq \mathsf{AInit} \mathbin{⨾} \tau^*_\mathsf{A} \tag{14}$$

$$\mathsf{CState}\uparrow \lhd \tau^{*|}_\mathsf{C} \mathbin{⨾} \mathsf{CFin} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AState}\uparrow \lhd \tau^{*|}_\mathsf{A} \mathbin{⨾} \mathsf{AFin} \tag{15}$$

$$\mathsf{CState}\uparrow \subseteq \mathrm{dom}(\mathsf{T} \rhd \mathsf{AState}\uparrow) \tag{16}$$

and for matching operations $\mathsf{AOp}$ and $\mathsf{COp}$:

$$\mathrm{dom}(\mathsf{T} \rhd \mathrm{liv}\, \overleftrightarrow{\mathsf{AOp}}) \mathbin{\lhd\!\!\!-} (\mathrm{liv}\, \overleftrightarrow{\mathsf{COp}} \mathbin{\lhd\!\!\!-} \overleftrightarrow{\mathsf{COp}}) \mathbin{⨾} \mathsf{T} \subseteq \mathsf{T} \mathbin{⨾} (\mathrm{liv}\, \overleftrightarrow{\mathsf{AOp}} \mathbin{\lhd\!\!\!-} \overleftrightarrow{\mathsf{AOp}}) \tag{17}$$

$$\mathrm{liv}\, \overleftrightarrow{\mathsf{COp}} \subseteq \mathrm{dom}(\mathsf{T} \rhd \mathrm{liv}\, \overleftrightarrow{\mathsf{AOp}}) \tag{18}$$

$$\overline{\mathrm{dom}\, \overleftrightarrow{\mathsf{COp}}} \subseteq \mathrm{dom}(\mathsf{T} \mathbin{\rhd\!\!\!\!-} \mathrm{dom}\, \overleftrightarrow{\mathsf{AOp}}) \tag{19}$$

We can prove a stronger theorem about closure of simulation under internal evolution in this case:

**Theorem 9 (Upward simulation closed under internal evolution).**
If the relation $\mathsf{T} \subseteq \mathsf{CState} \times \mathsf{AState}$ is an upward simulation between program controlled basic data types with internal operations $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in I}, \tau_\mathsf{A}, \mathsf{AFin})$ and $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in I}, \tau_\mathsf{C}, \mathsf{CFin})$ then $\tau_\mathsf{C}^* \mathbin{⨾} \mathsf{T} \mathbin{⨾} \tau_\mathsf{A}^*$ is also an upward simulation between $\mathsf{A}$ and $\mathsf{C}$.

**Proof** By showing that (14) to (19) above imply the same conditions with $\tau_\mathsf{C}^* \mathbin{⨾} \mathsf{T} \mathbin{⨾} \tau_\mathsf{A}^*$ substituted for $\mathsf{T}$. □

The initialisation condition can be simplified using Theorem 9, clearly $\tau_\mathsf{C}^* \mathbin{⨾} \mathsf{T} = \mathsf{T}$. Using totality of $\mathsf{T}$ and the blocking condition, the correctness condition can be simplified to remove two anti-restrictions.

In general, the conditions are not as clear-cut concerning divergent states as the downward simulation ones, where finalisation applicability ensures that the simulation links divergent states to divergent states only. Here, the finalisation conditions merely ensure that every abstract divergent state is linked to a concrete one, and vice versa. The blocking condition (19) also requires divergent concrete states to be linked to abstract states allowing the same operations. This is clearly a requirement inherited from the relational refinement theory that is redundant and irrelevant due to the chaotic interpretation of divergence represented in our semantics. Thus, using a general semantic theorem we restrict condition (19) to non-divergent states only; however, in those states it is dominated by the finalisation correctness condition (15). This is summarised, with some simplifications, in the following theorem.

**Theorem 10 (Upward simulation for data types with internal operations (blocking)).**
The relation $\mathsf{T}' \subseteq \mathsf{CState} \times \mathsf{AState}$ is an upward simulation between program controlled basic data types with internal operations $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in I}, \tau_\mathsf{A}, \mathsf{AFin})$ and $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in I}, \tau_\mathsf{C}, \mathsf{CFin})$ iff the following conditions hold for $\mathsf{T} == \tau_\mathsf{C}^* \mathbin{⨾} \mathsf{T}' \mathbin{⨾} \tau_\mathsf{A}^*$:

$$\mathsf{CInit} \mathbin{⨾} \mathsf{T} \subseteq \mathsf{AInit} \mathbin{⨾} \tau_\mathsf{A}^* \tag{20}$$

$$\mathsf{CState}\!\uparrow\; \lhd \tau_\mathsf{C}^{*|} \mathbin{⨾} \mathsf{Ref}_\mathsf{C} \subseteq (\mathsf{T} \lhd \mathsf{AState}\!\downarrow) \mathbin{⨾} \mathsf{Ref}_\mathsf{A} \tag{21}$$

$$\mathsf{CState}\!\uparrow\; \subseteq \operatorname{dom}(\mathsf{T} \rhd \mathsf{AState}\!\uparrow) \tag{22}$$

and for matching operations $\mathsf{AOp}$ and $\mathsf{COp}$:

$$\operatorname{dom}(\mathsf{T} \rhd \operatorname{liv} \overset{\leftrightarrow}{\mathsf{AOp}}) \lhd \tau_\mathsf{C}^* \mathbin{⨾} \mathsf{COp} \mathbin{⨾} \mathsf{T} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AOp} \mathbin{⨾} \tau_\mathsf{A}^* \tag{23}$$

$$\operatorname{liv} \overset{\leftrightarrow}{\mathsf{COp}} \subseteq \operatorname{dom}(\mathsf{T} \rhd \operatorname{liv} \mathsf{AOp}) \tag{24}$$

□

For specifications in Z, this leads to the following conditions. (See [4] for the essential steps in rephrasing the finalisation condition.)

**US.InitBlock.internal** $\forall\, CState';\ AState' \bullet T' \land CInit \Rightarrow AInit \mathbin{⨾} \tau_A^*$

**US.CorrBlock.internal** $\forall\, i : I;\ CState;\ CState';\ AState' \bullet$
$(\forall\, AState \bullet T \Rightarrow \neg \operatorname{liv} \overset{\leftrightarrow}{AOp_i}) \land COp_i \land T' \Rightarrow \exists\, AState \bullet T \land AOp_i \mathbin{⨾} \tau_A^*$

**US.FinRef.internal** $\forall\, CState;\ CState' \mid \neg CState\!\uparrow\; \land \tau_C^* \land CState'\!\downarrow\; \bullet$
$\exists\, AState \bullet T \land AState\!\downarrow\; \land \forall\, i : I \bullet \operatorname{pre} AOp_i \Rightarrow (\operatorname{pre} COp_i)'$

**US.DivStates** $\forall\, CState \bullet CState\!\uparrow\; \Rightarrow \exists\, AState \bullet T \land AState\!\uparrow$

**US.DivOp** $\forall\, i : I;\ CState \bullet \operatorname{liv} \overset{\leftrightarrow}{COp_i} \Rightarrow \exists\, AState \bullet T \land \operatorname{liv} AOp_i$

Again note that in the absence of internal operations, these conditions collapse to those in Section 3.3.

## 5.6. The non-blocking approach

Having considered refinement and simulation for data types with internal operations in the blocking approach in great detail, we now turn to the non-blocking approach, (as previously) not yet considering outputs. In that context, the semantics is considerably simpler: there are no refusals beyond those after divergence (the basic finalisation maps to a single global value representing a non-divergent run; its totalisation adds a representation of divergence). The embedding into process data types was already given above in Definition 12, the only difference being that an operation $\mathsf{Op}$ is embedded by $(\mathsf{B}, \mathsf{N}, \mathsf{D})$ where

$$\mathsf{B} == \varnothing \qquad\qquad \mathsf{D} == \text{liv}\,\overleftrightarrow{\mathsf{Op}} \cup \overline{\text{dom}\,\overleftrightarrow{\mathsf{Op}}}$$

As discussed in Section 3.4, in this context failures-divergences refinement is equivalent to traces-divergences refinement. Moreover, the trace sets for all ADTs with the same alphabet are identical. We thus need to look at divergences only. For a definition of (non-blocking) failures-divergences semantics analogous to Definition 15, Lemma 1 holds and forms the essence of the correctness proof of this embedding.

### 5.6.1. *Downward simulation*

We now extract the simulation rules for the non-blocking approach in a way similar to that in Sections 5.5.1 and 5.5.2. Instantiating Theorem 4 with the embedding of Definition 12 initially gives:

$$\mathsf{CInit} \mathbin{⨾} \tau_{\mathsf{C}}^{*} \subseteq \mathsf{AInit} \mathbin{⨾} \tau_{\mathsf{A}}^{*} \mathbin{⨾} \mathsf{R} \tag{25}$$

$$\mathsf{R} \mathbin{⨾} \mathsf{CState}\!\uparrow\; \lhd \tau_{\mathsf{C}}^{*|} \mathbin{⨾} \mathsf{CFin} \subseteq \mathsf{AState}\!\uparrow\; \lhd \tau_{\mathsf{A}}^{*|} \mathbin{⨾} \mathsf{AFin} \tag{26}$$

$$(\mathsf{AState}\!\uparrow\; \lhd \mathsf{R}) = (\mathsf{R} \mathbin{\ntriangleright} \mathsf{CState}\!\uparrow\;) \tag{27}$$

and for matching operations $\mathsf{AOp}$ and $\mathsf{COp}$:

$$(\text{liv}\,\overleftrightarrow{\mathsf{AOp}} \cup \overline{\text{dom}\,\overleftrightarrow{\mathsf{AOp}}}) \lhd \mathsf{R} \mathbin{⨾} (\text{liv}\,\overleftrightarrow{\mathsf{COp}}) \ntriangleleft \overleftrightarrow{\mathsf{COp}} \subseteq (\text{liv}\,\overleftrightarrow{\mathsf{AOp}} \ntriangleleft \overleftrightarrow{\mathsf{AOp}}) \mathbin{⨾} \mathsf{R} \tag{28}$$

$$\text{dom}(\mathsf{R} \rhd (\text{liv}\,\overleftrightarrow{\mathsf{COp}} \cup \overline{\text{dom}\,\overleftrightarrow{\mathsf{COp}}})) \subseteq \text{liv}\,\overleftrightarrow{\mathsf{AOp}} \cup \overline{\text{dom}\,\overleftrightarrow{\mathsf{AOp}}} \tag{29}$$

$$\text{dom}(\mathsf{R} \rhd \varnothing) \subseteq \varnothing \tag{30}$$

These can be simplified, although there is no theorem analogous to Theorem 7 (simulations closed under $\tau$) – this is due to the fact that properties such as

$$\mathsf{s} \in \text{dom}\,\overleftrightarrow{\mathsf{Op}} \wedge (\mathsf{s}, \mathsf{s}') \in \tau^{*} \Rightarrow \mathsf{s}' \in \text{dom}\,\overleftrightarrow{\mathsf{Op}}$$

do not hold in general. The finalisation condition (26) is implied by finalisation applicability (27) for the trivial finalisation. The blocking condition (30) is obviously satisfied. Two domain restrictions in (28) can be removed, one due to preservation of divergence in (29). All in all, this establishes the following.

**Theorem 11 (Downward simulation for data types with internal operations (non-blocking)).** The relation $\mathsf{R} \subseteq \mathsf{AState} \times \mathsf{CState}$ is a downward simulation between program controlled basic data types with internal operations $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in I}, \tau_{\mathsf{A}}, \mathsf{AFin})$ and $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in I}, \tau_{\mathsf{C}}, \mathsf{CFin})$ iff the following conditions hold:

$$\mathsf{CInit} \mathbin{⨾} \tau_{\mathsf{C}}^{*} \subseteq \mathsf{AInit} \mathbin{⨾} \tau_{\mathsf{A}}^{*} \mathbin{⨾} \mathsf{R} \tag{31}$$

$$(\mathsf{AState}\!\uparrow\; \lhd \mathsf{R}) = (\mathsf{R} \rhd \mathsf{CState}\!\uparrow\;) \tag{32}$$

and for matching operations $\mathsf{AOp}$ and $\mathsf{COp}$:

$$(\mathrm{liv}\,\overleftrightarrow{\mathsf{AOp}} \cup \overline{\mathrm{dom}\,\overleftrightarrow{\mathsf{AOp}}}) \lhd \mathsf{R} \mathbin{⨾} \overleftrightarrow{\mathsf{COp}} \;\subseteq\; \overleftrightarrow{\mathsf{AOp}} \mathbin{⨾} \mathsf{R} \tag{33}$$

$$\mathrm{dom}(\mathsf{R} \rhd (\mathrm{liv}\,\overleftrightarrow{\mathsf{COp}} \cup \overline{\mathrm{dom}\,\overleftrightarrow{\mathsf{COp}}})) \;\subseteq\; \mathrm{liv}\,\overleftrightarrow{\mathsf{AOp}} \cup \overline{\mathrm{dom}\,\overleftrightarrow{\mathsf{AOp}}} \tag{34}$$

□

For specifications in Z this leads to the following conditions (using $\Delta CState$ for $CState;\ CState'$):

**DS.InitNonBlock.internal** $\forall CState' \bullet CInit \mathbin{⨾} \tau_C^* \Rightarrow \exists AState' \bullet AInit \mathbin{⨾} \tau_A^* \wedge R'$
**DS.AppNonBlock.internal**
$\forall CState;\ AState;\ i : I \bullet R \wedge \mathrm{pre}\,\overleftrightarrow{AOp_i} \wedge \neg\,\mathrm{liv}\,\overleftrightarrow{AOp_i} \Rightarrow \mathrm{pre}\,\overleftrightarrow{COp_i} \wedge \neg\,\mathrm{liv}\,\overleftrightarrow{COp_i}$
**DS.CorrNonBlock.internal**
$\forall i : I;\ AState;\ \Delta CState \bullet \neg\,\mathrm{liv}\,\overleftrightarrow{AOp_i} \wedge \mathrm{pre}\,\overleftrightarrow{AOp_i} \wedge R \wedge \overleftrightarrow{COp_i} \Rightarrow \exists AState' \bullet \overleftrightarrow{AOp_i} \wedge R'$
**DS.DivStates** $\forall AState;\ CState \mid R \bullet AState\!\uparrow\ \Leftrightarrow CState\!\uparrow$

### *5.6.2. Upward simulation*

Again instantiating the upward simulation theorem for process data types with the non-blocking embedding into process data types, we obtain the following, after some small simplifications.

**Theorem 12 (Upward simulation for data types with internal operations (non-blocking)).**
The relation $\mathsf{T} \subseteq \mathsf{CState} \times \mathsf{AState}$ is an upward simulation between program controlled basic data types with internal operations $\mathsf{A} = (\mathsf{AState}, \mathsf{AInit}, \{\mathsf{AOp}_i\}_{i \in I}, \tau_{\mathsf{A}}, \mathsf{AFin})$ and $\mathsf{C} = (\mathsf{CState}, \mathsf{CInit}, \{\mathsf{COp}_i\}_{i \in I}, \tau_{\mathsf{C}}, \mathsf{CFin})$ iff the following conditions hold:

$$\mathsf{CInit} \mathbin{⨾} \tau_{\mathsf{C}}^* \mathbin{⨾} \mathsf{T} \;\subseteq\; \mathsf{AInit} \mathbin{⨾} \tau_{\mathsf{A}}^* \tag{35}$$

$$\overline{\mathsf{CState}\!\uparrow} \;\subseteq\; \mathrm{dom}(\mathsf{T} \mathbin{\overline{\rhd}} \mathsf{AState}\!\uparrow\,) \tag{36}$$

$$\mathsf{CState}\!\uparrow \;\subseteq\; \mathrm{dom}(\mathsf{T} \rhd \mathsf{AState}\!\uparrow\,) \tag{37}$$

and for matching operations $\mathsf{AOp}$ and $\mathsf{COp}$, using $\mathrm{div}\,\mathsf{Op} == \mathrm{liv}\,\overleftrightarrow{\mathsf{Op}} \cup \overline{\mathrm{dom}\,\overleftrightarrow{\mathsf{Op}}}$:

$$\mathrm{dom}(\mathsf{T} \rhd \mathrm{div}\,\mathsf{AOp}) \lhd \overleftrightarrow{\mathsf{COp}} \mathbin{⨾} \mathsf{T} \;\subseteq\; \mathsf{T} \mathbin{⨾} \overleftrightarrow{\mathsf{AOp}} \tag{38}$$

$$\mathrm{div}\,\mathsf{COp} \;\subseteq\; \mathrm{dom}(\mathsf{T} \rhd \mathrm{div}\,\mathsf{AOp}) \tag{39}$$

□

For specifications in Z, this leads to the following conditions:

**US.InitNonBlock.internal** $\forall CState';\ AState' \bullet CInit \mathbin{⨾} \tau_C^* \wedge T' \Rightarrow AInit \mathbin{⨾} \tau_A^*$
**US.AppDivOp** $\forall i : I;\ CState \bullet \mathrm{div}\,COp_i \Rightarrow \exists AState \bullet T \wedge \mathrm{div}\,AOp_i$
**US.CorrNonBlock.internal** $\forall i : I;\ \Delta CState;\ AState' \bullet$
$(\forall AState \bullet T \Rightarrow \neg\,\mathrm{div}\,AOp_i) \wedge \overleftrightarrow{COp_i} \wedge T' \Rightarrow \exists AState \bullet T \wedge \overleftrightarrow{AOp_i}$
**US.DivStates** $\forall CState \bullet CState\!\uparrow\ \Rightarrow \exists AState \bullet T \wedge AState\!\uparrow$
**US.NonDivStates** $\forall CState \mid \neg CState\!\uparrow\ \bullet \exists AState \bullet T \wedge \neg AState\!\uparrow$

using the analogous definition of div $Op$, etc.

The downward and upward conditions derived in this section extend those discussed in Section 3.3 by adding internal evolution. As discussed in Section 3.4 no additional conditions are necessary in order to achieve equivalence with failures-divergences refinement – that only becomes necessary when we add outputs to the model, which we do now for both blocking and non-blocking models.

## 6. Outputs

In this section we enhance the results of the previous section by also considering data types with outputs. The course of our argument will be:

- The addition of outputs as characterised in data types with output embeddings has a very localised effect on the refinement conditions. This is explored in Section 6.1.
- The addition of outputs leads to refusals, even in the non-blocking approach, and even when outputs are deterministic; as a consequence more stringent simulation conditions derive from the finalisation observing refusals. This is explored in Sections 6.2 and 6.3.

### 6.1. Refinement conditions for output embeddings

Embedding, for example, a Z state-and-operations specification with state *State* and output type *Output* into a relational framework normally involves the creation of what we have called an output embedding, where the global and local state contain an output sequence as well as the "real" state, see Definition 6.

The standard derivation of Z refinement rules from such embeddings is given in [46] and [17, Section 4.5]. From the latter, it is clear that the properties contained in Definition 6 suffice to derive refinement conditions which vary only in small details from those without outputs:

- the initialisation and finalisation applicability conditions are unaffected;
- all quantifications over after-states in the operation conditions, including in the definition of the precondition pre, are extended with the same quantification over the operation's output *Output*.

This result is obtained using a retrieve relation that is the identity on the output sequence – this is enforced by the particular finalisation which prevents change of output (type) in refinement (IO-refinement [17, Chapter 10] removes this restriction).

Consider a process data type $\mathsf{D}$, its reduction $\mathsf{D_r}$ and its embedding in total data types $\mathsf{D_e}$, where $\mathsf{D_r}$ is an output embedding. First, clearly $\mathsf{D_e}$ is *not* an output embedding: its global state will be $(\mathsf{GB} \times \mathrm{seq}\, Output)_{\perp,\omega}$ when $\mathsf{D_r}$ has $\mathsf{GB} \times \mathrm{seq}\, Output$. However, the construction of $\mathsf{D_e}$ still guarantees (in both the blocking and non-blocking case) the crucial property: an operation's result is independent of previously produced outputs. A state $(\mathsf{ls}, \mathsf{os})$ being in the domain of an operation only depends on $\mathsf{ls}$, due to the last condition of Definition 6, and thus $\mathsf{ls}$ alone determines whether an operation will block or diverge for that reason.

Some obvious restrictions on the internal operation $\tau$ are also required to ensure this. First, $\tau$ cannot produce outputs (as this would make its occurrence visible), and second, it must be independent of previous outputs just like normal operations (cf. the last condition of Definition 6). So in the context of data types with outputs, we will also require:

$$((\mathsf{ls}, \mathsf{os}), (\mathsf{ls2}, \mathsf{os2})) \in \tau \Rightarrow \mathsf{os2} = \mathsf{os} \tag{40}$$

$$((\mathsf{ls}, \mathsf{os}), (\mathsf{ls2}, \mathsf{os})) \in \tau \Rightarrow ((\mathsf{ls}, \mathsf{os2}), (\mathsf{ls2}, \mathsf{os2})) \in \tau \tag{41}$$

This guarantees that the occurrence of livelock is independent of previous outputs, and thus the previous output sequence does not affect outcomes of operations in *any* of the three parts ($\mathsf{N}, \mathsf{B}, \mathsf{D}$) of an operation. Consequently, the derivations in the previous section can be adapted with minor modifications as previously.

From this point on, we will concentrate on deriving the Z rules (which are explicit about outputs) rather than the relational ones, taking the rules derived in the previous section as our basis. Note that the embedding of Z operations into relational ones ensures that conditions in Definition 6 and properties (40) to (41) are satisfied. Additional consequences from output refusals in finalisations will be outlined in the next subsections.

## 6.2. Outputs in the blocking approach

Recall that the inclusion of outputs changes the notion of an event – rather than just an index $i : I$, it now also includes an output value. Traces (including divergences) and refusal sets will have these events as their elements. Definition 8 provides two possible refusal finalisations in the blocking approach; here we only consider the *demonic* view of output, where the system is in charge of selecting an output value and consequently output values may be refused if alternative output values are possible.

The proof that the relational embedding using this finalisation correctly represent the failures-divergences semantics characterised by demonic output refusals proceeds similarly to the proof of Theorem 6, with the following modifications:

- Traces (divergences) and refusals now have (operation, output) pairs as their elements.
- Lemma 1 (characterisation of divergences) needs to be strengthened, as the output generated up to the point of divergence needs to be recovered before the state degenerates to $\omega$.
- Lemma 2 (correctness of observation of a successful run) has no such issues, as the output sequence forms part of the global state value generated.
- Lemma 3 (consistency of $\perp$ with refusals) is still essentially correct (generalising the refused operation to refusing all possible outputs for it), and allows recovery of the output sequence at the point of blocking.
- The proof of the main theorem is then mostly identical, using modified lemmas as suggested above.

### *6.2.1. Downward simulation*

As indicated above, the initialisation conditions and conditions for operations remain unchanged from Section 5.5.1. Finalisation applicability is unchanged: we still finalise only in non-divergent states. What is also unchanged is that in any such state, we observe all possible refusals in all stable states reachable from it by (necessarily finite) internal evolution. Moreover, we can assume that the retrieve relation is closed under composition with $\underline{\tau}_{\mathsf{C}}^{*}$ (Theorem 7). Using similar reasoning as for the corresponding condition in Section 5.5.1, we end up with the proof obligation:

$$\forall\, a : AState;\ c' : CState;\ E \bullet (a, c') \in R \land c' \in CState\!\downarrow\ \land FcondD(c', E)$$
$$\Rightarrow \exists\, a' : AState \bullet (a, a') \in \tau_A^{*|} \land FcondD(a', E)$$

Unlike in Section 5.5.1, this is *not* implied by the other conditions. This is clear from the following counterexample.

$AState$
$a : a0 \mid a1 \mid a2$

$CState$
$c : \{c0\}$

$AInit$
$AState'$
---
$a' = a0$

$CInit$
$CState'$

$\tau_A$
$\Delta AState$
---
$a = a0 \wedge a' \in \{a1, a2\}$

$P_A$
$\Delta AState;\ x! : \{1, 2\}$
---
$(a = a1 \wedge x! = 1) \vee (a = a2 \wedge x! = 2)$
$a' = a0$

$P_C$
$\Delta CState;\ x! : \{1, 2\}$

$Q_A == P_A$

$Q_C == P_C$

The concrete data type can refuse $\{P!1, Q!2\}$ after the empty trace; this is not possible in the abstract data type. Thus, it is not a refinement, however, it satisfies all downward simulation conditions except the finalisation condition.[12]

The remaining finalisation condition looks cumbersome to check (quantifying over all sets of events that a concrete state might refuse); however two further simplifications are possible:

- only sets $E$ that are maximal in the concrete state need to be considered; downward closedness of refusals in the linked abstract state then ensures that the property is also satisfied for subsets;
- only events refused because of the availability of alternative outputs need to be considered; operations whose precondition does not hold in the concrete state must be refused in any linked abstract state, due to the blocking condition.

Maximal refusals are most easily characterised by their complements, which select a single output value for each enabled operation (and refuse all other output values, whether possible or impossible), i.e., they are partial functions from $I$ to $Output$. $Maxsim$ is a schema operation, in this case effectively a predicate on states, parameterised by such a partial function (and implicitly by the state schema).

$Sim == I \pfun Output$
$Maxsim(E) == \forall i : I \setminus \operatorname{dom} E \bullet \neg \operatorname{pre} Op_i \wedge \forall i : \operatorname{dom} E \bullet \exists State';\ Output \bullet Op_i \wedge \theta Output = E(i)$

Thus, in addition to the conditions from Section 5.5.1, we require here

**DS.FinDemBlock.internal**

$$\begin{array}{l} \forall AState;\ CState;\ E \mid R \wedge CState\downarrow\ \wedge Maxsim(E) \bullet \\ \quad \exists AState' \bullet \tau_A^* \wedge AState'\downarrow\ \wedge \forall i : \operatorname{dom} E \bullet (\exists (AOp_i)' \bullet \theta Output' = E(i)) \vee \neg \exists (AOp_i)' \end{array}$$

where the consequent ensures that for each operation that is enabled in the concrete state, some stable abstract state connected by $R \mathbin{⨾} \tau_A^*$ either selects the same output, or disables the operation (leading to a superset of refusals in either case).

[12] Taking a CSP view, let $P_i := p!i \rightarrow stop$ etc, and writing $(\tau \rightarrow A)\Box(\tau \rightarrow B)$ as $A \sqcap B$, this example is $(P_1 \Box Q_1) \sqcap (P_2 \Box Q_2) \not\sqsubseteq (P_1 \sqcap P_2) \Box (Q_1 \sqcap Q_2)$.

In the counterexample above, this condition fails on the maximal refusal set $\{P!1, Q!2\}$ represented by its complement $\{P!2, Q!1\}$. There is no abstract state which, for each of these events, either allows it or completely disallows the operation ($P$ or $Q$).

In Section 3.4 we discussed how, in the blocking model with input/output, we needed the condition **US.FinDem** to correctly link up refusal sets due to outputs. The example above has shown that, in the presence of internal operations, a similar extra condition is needed for downward simulations, and this we have called **DS.FinDemBlock.internal**.

### *6.2.2. Upward simulation*

The reasoning here is similar to the corresponding case without outputs. Again, for a concrete non-divergent state $c$, we need to find a linked abstract state for each stable state $c'$ reachable from $c$ by internal evolution. However, as in this model such a state $c'$ does not necessarily have a single maximal refusal set, we need to find a (possibly different) linked abstract state for each maximal refusal set in each such $c'$.

**US.FinDemBlock.internal**

$$\begin{array}{l} \forall\, E : Sim;\ \Delta CState \mid \neg CState\uparrow\ \wedge \tau_C^* \wedge CState'\downarrow\ \wedge (Maxsim(E))' \bullet \\ \qquad \exists\, AState, F : Sim \bullet\ T \wedge AState\downarrow\ \wedge F \subseteq E \wedge Maxsim(F) \end{array}$$

In the absence of internal operations this, of course, collapses to **US.FinDem**.

## 6.3. Outputs in the non-blocking approach

In the non-blocking approach with demonic outputs, it may at first seem surprising that there are refusals (other than those after divergence), even in states where operations are not enabled (and not blocked either!) First, we still expect output values to be determined "by the system", and thus output values will still be refused if other values are possible. Moreover, consider an operation that is not enabled (in a particular state), and thus leads to divergence. If none of the possible output values of that operation is refused, it also means that defining the operation to be enabled and have a particular output (or any choice of outputs) will not be a valid refinement, as it introduces new refusals. Thus, where an operation is not enabled in a particular state, we will allow any set of output values to be refused, except for the full set. In particular, this means that an operation whose output is from a singleton set will produce no refusals. This is consistent with the view that having no output, or having an output from a singleton set are isomorphic.

The resulting definition of refusals is then given by modifying the predicate $FcondD$ in Definition 8 as follows.

$$\mathsf{Fin} == \{State;\ os : \mathrm{seq}\, Output;\ E : \mathbb{P}(I \times Output) \mid Fcond \bullet (os, \theta State) \mapsto (os, E)\}$$

with $Fcond = FcondND(\theta State, E)$ where

$$FcondND(s, E) == E \subseteq \{(i, out) \mid \begin{array}{l} \exists\, State';\ Output \bullet \theta State = s\ \wedge \\ (\mathrm{pre}\, Op_i \Rightarrow Op_i) \wedge \theta Output \neq out \wedge (i, \theta Output) \notin E \end{array} \} \qquad (42)$$

Thus, the first disjunct from Definition 8 defining refusals when the operation is not applicable is dropped. The operation $Op_i$ is replaced by (effectively a totalisation) pre $Op_i \Rightarrow Op_i$ which allows an arbitrary result outside the precondition.

### 6.3.1. Downward simulation

The example in Section 6.2.1 also shows why the rules from Theorem 11 do not suffice in the case of demonic output refusals. (It is not a refinement, but it satisfies all the conditions of Theorem 11. However, the finalisation condition fails.)

Assuming the other conditions, the finalisation condition in this case reduces to

$$\begin{array}{l}\forall\, a : AState;\ c, c' : CState;\ E \bullet (a, c) \in R \wedge (c, c') \in \tau_C^* \wedge c \notin CState\!\uparrow\ \wedge c' \in CState\!\downarrow\ \wedge FcondND(c', E)\\ \Rightarrow \exists\, a' : AState \bullet (a, a') \in \tau_A^* \wedge a' \in AState\!\downarrow\ \wedge FcondND(a', E)\end{array}$$

Compared to the condition in Section 6.2.1, we require consideration of $\tau_C^*$ as well, as we cannot assume that $R$ is closed under composition with $\underline{\tau}_C^*$. As usual, it suffices to consider only maximal refusal sets in particular states (here: $c'$), and these can be represented by their complements. These now select an output for *every* operation (enabled or not), and thus they are *total* functions. This is characterised by:

$$Maxsimtot(E) \ == \ \mathrm{dom}\, E = I \wedge \forall\, i : I \bullet \mathrm{pre}\, Op_i \Rightarrow \exists\, State';\ Output \bullet Op_i \wedge \theta Output = E(i)$$

This leads to the following refinement condition.

**DS.FinDemNonBlock.internal**

$$\begin{array}{l}\forall\, AState;\ E;\ CState;\ CState' \mid R \wedge \neg CState\!\uparrow\ \wedge \tau_C^* \wedge CState'\!\downarrow\ \wedge (Maxsimtot(E))' \bullet\\ \qquad \exists\, AState' \bullet \tau_A^* \wedge AState'\!\downarrow\ \wedge (Maxsimtot(E))'\end{array}$$

As $E$ is a total function from $I$ to $Output$, it necessarily also represents a maximal set of simultaneously enabled events in $A$.

### 6.3.2. Upward simulation

For upward simulations we require the conditions given in Section 5.6.2 and, additionally, the conditions represented by the finalisation $\mathsf{CFin} \subseteq \mathsf{T} \mathbin{;} \mathsf{AFin}$. This leads to the requirement that

$$\begin{array}{l}\forall\, c, c' : CState;\ E \bullet c \notin CState\!\uparrow\ \wedge (c, c') \in \tau_C^* \wedge c' \in CState\!\downarrow\ \wedge FcondND(c', E)\\ \Rightarrow \exists\, a, a' : AState \bullet (c, a) \in T \wedge (a, a') \in \tau_A^* \wedge a' \in AState\!\downarrow\ \wedge FcondND(a', E)\end{array}$$

where $FcondND$ is defined in (42) above. Concentrating on maximal refusal sets in $c'$ and rephrasing that in terms of their complements gives:

**US.FinDemNonBlock.internal** $\forall\, \Delta CState;\ E : Sim \mid \neg CState\!\uparrow\ \wedge \tau_C^* \wedge CState'\!\downarrow\ \wedge (Maxsimtot(E))' \bullet$
$\quad \exists\, \Delta AState \bullet TAState'\!\downarrow\ \wedge \tau_S^* \wedge (Maxsimtot(E))'$

In this non-blocking model, refusals arise solely from the presence of outputs, since the internal choice of an observable aspect means refusals can occur if the environment was only prepared to accept a different value. To understand the requirement, consider the following scenario, in a simpler context without any internal operations.

In the first there is just one operation in $C$ without input or output, and none in $A$ (see Figure 5.). The non-blocking totalisation of $A$ (given as a CSP process) is

$$Op \rightarrow \mathrm{div}$$

In the initial state, the refusals of $A$ and $C$ are thus the same (i.e., none). This is consistent with the construction of $E$ and $FcondND$, since in both $A$ and $C$ we cannot find different output values occurring, thus $E = \varnothing$. The finalisation condition thus does not impose any further constraint on upward simulations in a model without input or output, since there are no refusals.

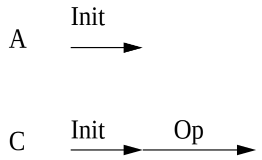


**Fig. 5.** Refusals in the non-blocking model

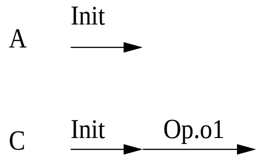


**Fig. 6.** Refusals in the non-blocking model - with outputs

However, now consider a model with input and output. Consider $A$ and $C$ with one operation that can potentially output $o_1$ or $o_2$. Adapting the above example, we derive that in Figure 6. The non-blocking totalisation of $A$ now has an internal choice of possible output values

$$(Op.o_1 \rightarrow \text{div}) \sqcap (Op.o_2 \rightarrow \text{div})$$

Thus refusals of $A$ after the empty trace are $\{Op.o_1\}$ and $\{Op.o_2\}$, whereas those of $C$ are just $\{Op.o_2\}$.

The presence of non-determinism in the specification, together with refusals due to outputs complicates the finalisation requirement. In particular, if an operation has a non-deterministic output at a given state, this can, in a refinement, be transferred to non-determinism prior to the operation invocation. This means the subsetting of refusals can be split across different states. For example, in Figure 7 the refusals initially in $C$ are $\{Op.o_1\}$ and $\{Op.o_2\}$ but not $\{Op.o_1, Op.o_2\}$. The condition $\forall\, CState;\ E \bullet FcondD(\theta CState, E) \Rightarrow \exists\, AState \bullet T \wedge FcondD(\theta AState, E)$ allows the chosen $AState$ for $E = \{Op.o_1\}$ to be $s = 2$ but that for $E = \{Op.o_2\}$ to be $s = 1$.

## 7. Conclusions

In this paper we have formed a general framework for reasoning about relational and process refinement. To do so we started out with an abstract data type specified in the usual Z "states-and-operations" style [43] but with some additional syntax to make the ADT structure explicit [17]. We then defined a particular process semantics for ADTs, and embedded an ADT in a relational data type in one of several ways, depending on the notion of observation. This enabled us to show that relational refinement of the embedded ADT was

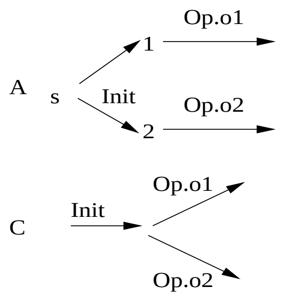


**Fig. 7.** Non-determinism and refusals inclusion

equivalent to a refinement of the process semantics. As a consequence, simulation rules on that particular relational embedding can then be used to verify the process refinement relation.

As we mentioned in the introduction similar approaches have been applied previously:

- to derive the usual Z refinement where the "process" semantics is not a concurrent one at all, observing only output values, with the "precondition" interpretation of operations – i.e., applying an undefined operation leads to divergence [46, 17];
- to derive refinement for Z with the "guard" interpretation, i.e., applying an undefined operation is impossible [9, 17];
- to define failures refinement for Z, in both the "guard" and "precondition" interpretations, observing refusals and outputs [4, 18, 40];
- to define readiness refinement for Z, using a version of relational refinement for partial relations [18];
- to define failures-divergences refinement for Z ADTs, in both interpretations, allowing internal operations but not outputs [19].

As noted in [32, 3], the consideration of internal operations [19] once again brings home that the two different "erroneous" behaviours, often appearing as different interpretations of partiality in operations, can – and often need to – co-exist. The "guards" or "blocking" interpretation leads to *deadlock*, i.e., applying an operation outside its domain leads to a special state which represents that a deadlock has occurred. If that deadlock is the only possible outcome for a particular trace (or series of choice points), this means that the trace is not allowed to occur, and thus this deadlock may not be removed in refinement. The "pre-condition" (or "non-blocking", or "contract") interpretation introduces *divergence*: an operation applied outside its domain is not contracted to produce any particular result, and may therefore produce any result including deadlock, or "worse". Typically one would take this "any result" view as underspecification, and allow for it to be removed in refinement. However, internal operations (at least in the CSP interpretation) bring in the possibility of divergence through livelock, independently of how partial operations are interpreted. In addition, we saw that outputs led to refusals even when we assume the non-blocking approach.

These considerations led to the definition of a relational datatype (which we called a *process data type*) with explicit characterisations of "normal" behaviour, divergence, and blocking, and observation of refusals at finalisation. Simulation rules (Sections 4.1 and 4.2) were derived for this generalised scheme "once and for all". We then subsequently showed how process data types could be given a number of different instantiations. The key result is the instantiation given in Section 5, which showed how a failures-divergences interpretation could be given for process data types. The correctness of the embedding was proved and simulation conditions derived which highlighted the role of divergence.

The main line of development that leads to Theorems 6, 8 and 10 has been formally verified using the KIV theorem prover [35]. Related theories verified previously include our earlier work on weak refinement ([20], in [38]) and Cooper, Stepney and Woodcock's generalised Z refinement rules ([15], in [39]). In contrast to these, formal theories and proofs were done during the development, and not a posteriori given the finished work. Mechanisation required three weeks of work: one for setting up specifications, one for proving theorems and one for adapting to changes. It uncovered a few missing assumptions (such as the stronger induction hypothesis for Lemma 2 using $f'$ rather than $f$), and helped to shorten proofs, such as that the finalisation condition is implied by the blocking condition for Theorem 8. A Web presentation automatically generated from KIV specifications and proofs is available [37].

These results extend previous work in a number of directions, specifically divergence due to internal evolution and the inclusions of outputs. In the survey given in Section 3 we gave the simulation conditions necessary for the standard non-blocking model (which corresponds to traces-divergences refinement), the standard blocking model (which corresponds to singleton failures refinement), and for a refusals embedding in a blocking model which corresponds to failures-divergences refinement. Section 5.5 extended this latter model by adding in the potential of internal evolution (but without outputs). This enabled the systematic derivation of the extension of Josephs' rules [30] to include arbitrary internal behaviour, which we believe has not been published previously. Section 5.6 gave the simulation conditions for a refusals embedding in the non-blocking approach (again without outputs). The full generality is reached in Sections 6.2 and 6.3 when the additional conditions are given when both outputs and internal evolution are included in the models. The inclusion of outputs is, of course, what makes the models subtle in terms of refusal information, and one direction of further research would be to attempt to further simplify condition **US.FinDem** and the various other conditions in terms of the complements of maximal refusal sets.

We have concentrated on the demonic interpretation of outputs, as the model that is, perhaps, more common, and presents us with more subtleties than the angelic. The angelic has, however, been advocated in the context of integrating formal methods. For example, Treharne and Schneider argue in [41] that the angelic model is more natural when CSP controllers are combined with B components in integrations of CSP and B. In particular, the CSP acts as a non-discriminating controller which cannot block on the values that the B component provides.

It is also worth noting that the angelic model can be more discriminating than the demonic model, e.g.,

$$(a \to b!1 \to stop)\Box(a \to (b!2 \to stop \Box b!3 \to stop))$$

is demonically equivalent to

$$(a \to b!1 \to stop)\Box(a \to b!2 \to stop)\Box(a \to b!3 \to stop)$$

However, these processes are not equivalent angelically. It is left to future work to explore the consequences of angelic outputs and how they might be used, for example, in integrations of formal methods in ways which were not possible for the demonic model.

## Acknowledgements

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

## References

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

[2] R. Berghammer and H. Zierer. Relation algebraic semantics of deterministic and non-deterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.

[3] E.A. Boiten and W.-P. de Roever. Getting to the bottom of relational refinement: Relations and correctness, partial and total. In R. Berghammer and B. Möller, editors, *7th International Seminar on Relational Methods in Computer Science (RelMiCS 7)*, pages 82–88. University of Kiel, May 2003.

[4] E.A. Boiten 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.

[5] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1988.

[6] C. Bolton. *On the refinement of state-based and event-based models.* PhD thesis, University of Oxford, 2002.

[7] C. Bolton and J. Davies. Refinement in Object-Z and CSP. In M. Butler, L. Petre, and K. Sere, editors, *Integrated Formal Methods (IFM 2002)*, volume 2335 of *Lecture Notes in Computer Science*, pages 225–244. Springer-Verlag, 2002.

[8] C. Bolton and J. Davies. A singleton failures semantics for Communicating Sequential Processes. *Formal Aspects of Computing*, 18:181–210, 2006.

[9] C. Bolton, J. Davies, and J.C.P. Woodcock. On the refinement and simulation of data types and processes. In K. Araki, A. Galloway, and K. Taguchi, editors, *International Conference on Integrated Formal Methods 1999 (IFM'99)*, pages 273–292, York, July 1999. Springer.

[10] C. Bolton and G. Lowe. A hierarchy of failures-based models. In F. Corradini and U. Westmann, editors, *Proceedings of Express 2003: 10th International Workshop on Expressiveness in Concurrency.* Elsevier Science Publishers, 2003.

[11] C. Bolton and G. Lowe. A hierarchy of failures-based models: Theory and application. *Theoretical Computer Science*, 330(3):407–438, 2005.

[12] H. Bowman and R. Gomez. *Concurrency Theory: Calculi and Automata for Modelling Untimed and Timed Concurrent Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[13] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[14] S.D. Brookes and A.W. Roscoe. An improved failures model for communicating processes. In S.D. Brookes, A.W. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 281–305. Springer, 1985.

[15] D. Cooper, S. Stepney, and J. Woodcock. Derivation of Z refinement proof rules: forwards and backwards rules incorporating input/output refinement. Technical Report YCS-2002-347, University of York, December 2002. URL: http://www-users.cs.york.ac.uk/∼susan/bib/ss/z/zrules.htm.

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

[17] J. Derrick and E.A. Boiten. *Refinement in Z and Object-Z: Foundations and Advanced Applications*. FACIT. Springer Verlag, May 2001.

[18] J. Derrick and E.A. Boiten. Relational concurrent refinement. *Formal Aspects of Computing*, 15(1):182–214, November 2003.

[19] J. Derrick and E.A. Boiten. Relational concurrent refinement with internal operations. In B. Aichernig, E.A. Boiten, J. Derrick, and L. Groves, editors, *BCS-FACS Refinement Workshop*, volume 187 of *ENTCS*, pages 35–53, 2006.

[20] J. Derrick, E.A. Boiten, H. Bowman, and M.W.A. Steen. Specifying and refining internal operations in Z. *Formal Aspects of Computing*, 10:125–159, December 1998.

[21] M. Deutsch and M.C. Henson. An analysis of refinement in an abortive paradigm. *Formal Aspects of Computing*, 18(3):329–363, 2006.

[22] H. Doornbos. A relational model of programs without the restriction to Egli-Milner constructs. In E.-R. Olderog, editor, *PROCOMET '94*, pages 357–376. IFIP, 1994.

[23] S. Dunne and S. Conroy. Process refinement in B. In H. Treharne, S. King, M.C. Henson, and S. Schneider, editors, *ZB 2005: Formal Specification and Development in Z and B, 4th International Conference of B and Z Users*, volume 3455 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2005.

[24] C. Fischer. CSP-OZ - A combination of CSP and Object-Z. In H. Bowman and J. Derrick, editors, *Second IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*, pages 423–438. Chapman & Hall, July 1997.

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

[26] M. Hennessy and A. Ingólfsdóttir. A theory of communicating processes with value passing. *Inf. Comput.*, 107(2):202–236, 1993.

[27] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[28] C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall, 1998.

[29] He Jifeng. Process refinement. In J. McDermid, editor, *The Theory and Practice of Refinement*. Butterworths, 1989.

[30] M.B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.

[31] G. Leduc. *On the Role of Implementation Relations in the Design of Distributed Systems using LOTOS*. PhD thesis, University of Liège, Liège, Belgium, June 1991.

[32] R. Miarka, E.A. Boiten, and J. Derrick. Guards, preconditions and refinement in Z. In J.P. Bowen, S. Dunne, A. Galloway, and S. King, editors, *ZB2000: Formal Specification and Development in Z and B*, volume 1878 of *Lecture Notes in Computer Science*, pages 286–303. Springer-Verlag, September 2000.

[33] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[34] S. Reeves and D. Streader. State- and event-based refinement. Technical report, Department of Computer Science, University of Waikato, September 2006.

[35] W. Reif, G. Schellhorn, K. Stenzel, and M. Balser. Structured specifications and interactive proofs with KIV. In W. Bibel and P. Schmitt, editors, *Automated Deduction—A Basis for Applications*, volume II: Systems and Implementation Techniques, chapter 1: Interactive Theorem Proving, pages 13 – 39. Kluwer Academic Publishers, Dordrecht, 1998.

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

[37] G. Schellhorn. Web presentation of the KIV proofs of 'Relational Concurrent Refinement Part II: Internal Operations and Output', 2006. URL: http://www.informatik.uni-augsburg.de/swt/projects/Refinement/Web/CSPRef/.

[38] G. Schellhorn. ASM refinement and generalizations of forward simulation in data refinement: A comparison. *Theoretical Computer Science*, 336(2-3):403–435, May 2005.

[39] G. Schellhorn, H. Grandy, D. Haneberg, and W. Reif. The Mondex challenge: Machine checked proofs for an electronic purse. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *Formal Methods 2006, Proceedings*, volume 4085 of *LNCS*, pages 16–31. Springer, 2006.

[40] S. Schneider. Non-blocking data refinement and traces-divergences semantics, 2006. Personal communication.

[41] S. Schneider and H. Treharne. CSP Theorems for Communicating B Machines. *Formal Aspects of Computing*, 17(4):390 – 422, December 2005.

[42] G. Smith and J. Derrick. Abstract specification in Object-Z and CSP. In C. George and H. Miao, editors, *Formal Methods and Software Engineering*, volume 2495 of *Lecture Notes in Computer Science*, pages 108–119. Springer, November 2002.

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

[44] A. Valmari and M. Tienari. Compositional failure-based semantics models for basic LOTOS. *Formal Aspects of Computing*, 7(4):440–468, 1995.

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

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

[47] J.C.P. Woodcock and C.C. Morgan. Refinement of state-based concurrent systems. In D. Bjorner, C.A.R. Hoare, and H. Langmaack, editors, *VDM'90: VDM and Z!- Formal Methods in Software Development*, volume 428 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.