More Relational Concurrent Refinement: Traces and Partial Relations

John Derrick[1]

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

Eerke Boiten[2]

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

Abstract

Data refinement in a state-based language such as Z is defined using a relational model in terms of the behaviour of abstract programs. Downward and upward simulation conditions form a sound and jointly complete methodology to verify relational data refinements. On the other hand, refinement in a process algebra takes a number of different forms depending on the exact notion of observation chosen, which can include the events a system is prepared to accept or refuse.
In this paper we continue our program of deriving relational simulation conditions for process algebraic refinement by defining further embeddings into our relational model: traces, completed traces, failure traces and extension.

Keywords: Data refinement, Z, simulations, process algebraic refinement preorders.

1 Introduction

Motivated by both theoretical comparisons of refinement and integrations of specification languages there has been significant interest in relating differing models of relational data refinement with those arising in a concurrent context.

In a process algebra such as CSP [18,24] a system is defined in terms of actions (or events) which represent the *inter*actions between a system and its environment. The exact way in which the environment is allowed to interact with the system varies between different 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

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

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

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

sets for different processes. As defined, the obvious and cumbersome method of verifying that refinement holds is by evaluating such set inclusions, between large and potentially infinite sets. A survey of many prominent refinement relations is given in [27].

State-based systems provide an alternate view, whereby specifications are considered to define abstract data types (ADTs), consisting of an initialisation, a collection of operations and a finalisation. A program over an ADT is a sequential composition of these elements. Refinement is defined 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 C refines an ADT A if for every program and sequence of inputs, the outputs that C produces are outputs that A could also have produced. This definition of refinement quantifies over program behaviour and *simulations* have become the accepted approach to make verification of refinements tractable [13]. For a complete method, often two kinds of simulations are defined: downward and upward simulations.

Research on combining relational and concurrent refinement concentrated initially on providing joint semantics, and on identifying correspondences between variations of the relational models and concurrency semantics. In the latter category, see e.g. work by Bolton and Davies [7,8] and Reeves and Streader [23]. Our work on relational concurrent refinement started [5,14] from the powerful idea that the relational *finalisations* can encode the observations embedded in concurrency semantics. The relational simulation rules can then be used to extract simulations for concurrency. These provide a "canned induction" method of verifying concurrent refinement, by checking a fixed number of conditions for each possible action, rather than checking inclusion between potentially large sets. We derived simulation rules for failures-divergences refinement [14,5], including also outputs and internal operations [4], and for readiness refinement [14]. These were mostly based on the total relations model (as described below).

This paper continues the programme, by considering more concurrent refinement relations, many of them based on the *partial* relations model. The structure of this paper is simple. In Section 2 we provide the basic definitions and background. In Section 3 we provide the simulation rules for a number of process algebraic preorders, and we conclude in Section 4.

2 Background

This background section presents the standard refinement theory [15] for abstract data types in a relational setting. The relational model of data refinement where all operations are total, as described in the 1986 paper by He, Hoare and Sanders [17], traditionally received the most attention. The standard refinement theory of Z [28,15], for example, is based on this version of the theory. However, later publications by He and Hoare, in particular [16], dropped the restriction to total relations, and proved soundness and joint completeness of the same set of simulation rules in the more general case. De Roever and Engelhardt [13] also present the partial relations theory, without putting much emphasis on this aspect.

2.1 A partial relational model

As usual [15], a program (defined here as a sequence of operations) is given as a relation over a global state G, implemented using a local state State. 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.

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 (Data type)
A 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 (total or partial) relations on the set* State*;* Init *is a total relation from* G *to* State*;* Fin *is a total relation from* State *to* G*.* □

Insisting that Init and Fin be total merely records the facts that we can always start a program sequence (the extension to partial initialisations is trivial and uninteresting) and that we can always make an observation.

Definition 2 (Complete program)
A complete program *over a data type* $\mathsf{D} = (\mathsf{State}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i \in I}, \mathsf{Fin})$ *is an expression of the form* $\mathsf{Init} \fatsemi \mathsf{P} \fatsemi \mathsf{Fin}$*, where* P*, a relation over* State*, is a program over* $\{\mathsf{Op}_i\}_{i \in I}$*. Programs are finite sequences of operations. For a sequence* p *over* I*, and data type* D*,* p_D *denotes the complete program over* D *characterised by* p*. For example, if* $\mathsf{p} = \langle \mathsf{p}_1, ..., \mathsf{p}_\mathsf{n} \rangle$ *then* $\mathsf{p}_\mathsf{D} = \mathsf{Init} \fatsemi \mathsf{Op}_{\mathsf{p}_1} \fatsemi ... \fatsemi \mathsf{Op}_{\mathsf{p}_\mathsf{n}} \fatsemi \mathsf{Fin}$*.* □

As usual we assume that the data types are *conformal*, i.e., they use the same index set for the operations.

Definition 3 (Data refinement for partial relations)
For partial data types A *and* C*,* C refines A*, denoted* $\mathsf{A} \sqsubseteq_{data} \mathsf{C}$ *(dropping the subscript if the context is clear), iff for each finite sequence* p *over* I*,* $\mathsf{p}_\mathsf{C} \subseteq \mathsf{p}_\mathsf{A}$*.* □

Downward and *upward* simulations [13] form a sound and jointly complete [17,13] proof method for verifying refinements. 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 4 (Downward simulation)
Assume 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* R *from* AState *to* CState *satisfying*

$$\mathsf{CInit} \subseteq \mathsf{AInit} \fatsemi \mathsf{R}$$
$$\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin}$$
$$\forall\, i : I \bullet \mathsf{R} \fatsemi \mathsf{COp}_i \subseteq \mathsf{AOp}_i \fatsemi \mathsf{R}$$

If such a simulation exists, we also say that C *is a downward simulation of* A *and similarly for corresponding operations of* A *and* C. □

Any relational data types A and C in this paper are assumed to be defined as in the above definition (occasionally with extra conditions imposed).

Definition 5 (Upward simulation)
For data types A *and* C, *an* upward *simulation is a relation* T *from* CState *to* AState *such that*

$$\mathsf{CInit} \fatsemi \mathsf{T} \subseteq \mathsf{AInit}$$
$$\mathsf{CFin} \subseteq \mathsf{T} \fatsemi \mathsf{AFin}$$
$$\forall\, i : I \bullet \mathsf{COp}_i \fatsemi \mathsf{T} \subseteq \mathsf{T} \fatsemi \mathsf{AOp}_i$$

If such a simulation exists, we also say that C *is an upward simulation of* A *and similarly for corresponding operations of* A *and* C. □

2.2 Totalisations

In terms of the observations it makes, the partial relational model described above has limitations. The fact that a trace p is "impossible" is represented by its interpretation p_D being the empty set. This may be interpreted as this trace (or a prefix of it) leading to a guaranteed deadlock. However, the relations contain non-determinism, and depending how this is resolved during the computation, a particular trace may or may not deadlock. In the partial relational model, it is not immediately observable that deadlock is possible but *not* guaranteed – thus, the non-determinism is interpreted angelically: negative results are ignored when positive ones also exist. Richer observations can be obtained in one of two ways. In most of this paper we will do so by staying within the partial relations model, by observing *more* at finalisation. This gives extra information not just for a trace, but through universal quantification over all traces, also for all its prefixes. This will, e.g., turn out to be enough to characterise possible deadlock when refusals are observed, see Section 3.3. Another way of observing possible as opposed to certain error is through modelling it explicitly by *totalising* the relations first. Elsewhere, a so-called *non-blocking* totalisation is used; here we only use the *blocking* totalisation defined below. The blocking totalisation still encodes the intuition, also present in the partial relations model, that an operation cannot be applied outside its domain (the "guard"), by mapping such applications to an explicit error value $\perp$. The non-blocking view, in contrast, maps such applications to all possible values (including an "error" one), modelling the interpretation that outside the domain (the precondition) "anything" can happen, including unspecified error. For the trace-based semantics considered in this paper, the non-blocking view is less interesting, as in that model all traces are possible. The totalisations turn a partial relation on a set S into a total relation on a set $\mathsf{S}_\perp$, which is S extended with a distinguished value $\perp$ not in S.

Definition 6 (Blocking totalisation)
For a partial relation Op *on* State*, its blocking totalisation is a total relation on* $\mathsf{State}_\perp$*, defined by*

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

□

Characterisations of downward and upward simulations on these totalised relations can be simplified to remove any reference to $\perp$. This results in the standard definitions of downward and upward simulations for partial relations [15].

Definition 7 (Downward simulation for totalised relations)
Given data types A *and* C *where the operations may be partial. A* downward *simulation is a relation* R *from* AState *to* CState *satisfying, in the blocking model*

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

□

Definition 8 (Upward simulation for totalised relations)
For data types A *and* C *where the operations may be partial, an* upward *simulation is a relation* T *from* CState *to* AState *satisfying, in the blocking model*

$$\begin{array}{l}
\mathsf{CInit} \mathbin{⨾} \mathsf{T} \subseteq \mathsf{AInit} \\
\mathsf{CFin} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AFin} \\
\forall\, i : I \bullet \overline{\operatorname{dom} \mathsf{COp}_i} \subseteq \operatorname{dom}(\mathsf{T} \rhd \operatorname{dom} \mathsf{AOp}_i) \\
\forall\, i : I \bullet \mathsf{COp}_i \mathbin{⨾} \mathsf{T} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AOp}_i
\end{array}$$

□

The conditions imposed on all operations in Definitions 7 and 8 are called "applicability" and "correctness" in both cases.

Note, however, that the upward and downward simulations given above are *not* jointly complete for blocking refinement [3], which means that completeness needs to be proved separately.

2.3 Refinement in Z

The definition of refinement in a specification language such as Z is usually based on the totalised framework just given, sticking with the blocking variant. Specifically, 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 defined in terms of (the variables of) $State$ (its before-state) and $State'$ (its after-state). The initialisation is also expressed in terms of an after-state $State'$. In addition to this, operations can also consume inputs and produce outputs. Finalisation normally does not appear explicitly in the simulation rules as presented in Z for reasons we shall see later.

If specifications have inputs and outputs, these are included in both the global and local state of the relational embedding of a Z specification. See [15] for the full details on this – in this paper we only consider datatypes without inputs and outputs. In concurrent refinement relations, inputs add little complication; outputs particularly complicate refusals as described in [4].

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 are as follows:

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

Given these embeddings, we can translate the relational refinement conditions of downward simulations into refinement conditions for Z ADTs, where we note that the finalisation conditions are always satisfied in this Z interpretation.

Definition 9 (Standard downward simulation in Z)
Given Z data types $A = (AState, AInit, \{AOp_i\}_{i \in I})$ *and* $C = (CState, CInit, \{COp_i\}_{i \in I})$. *The relation* R *on* $AState \wedge CState$ *is a* downward simulation *from* A *to* C *in the blocking model if*

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

and for all $i : I$*:*

$$\begin{array}{l}
\forall\, AState;\ CState \bullet R \Rightarrow (\mathrm{pre}\, AOp_i \Leftrightarrow \mathrm{pre}\, COp_i) \\
\forall\, AState;\ CState;\ CState' \bullet R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i
\end{array}$$

□

Any Z data types A and C in this paper are assumed to be defined as in the above definition.

The translation of the upward simulation conditions is similar, however this time the finalisation produces a condition that the simulation is total on the concrete state.

Definition 10 (Standard upward simulation in Z)
For Z data types A *and* C*, the relation* T *on* $AState \wedge CState$ *is an* upward simulation *from* A *to* C *in the blocking model if*

$$\begin{array}{l}
\forall\, CState \bullet \exists\, AState \bullet T \\
\forall\, AState';\ CState' \bullet CInit \wedge T' \Rightarrow AInit
\end{array}$$

and for all $i : I$*:*

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

□

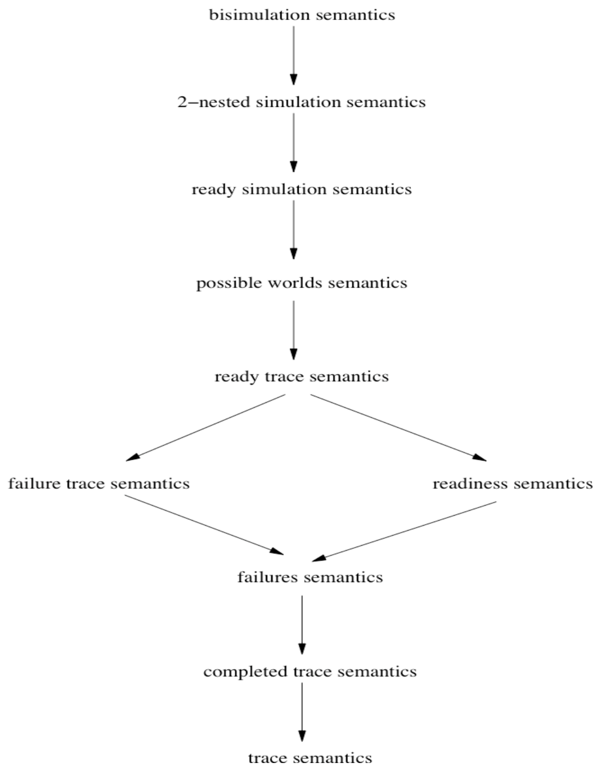

Fig. 1. The linear time - branching time spectrum [27]

3 Process algebraic based refinement

Process algebras [18,22,2] provide a means to describe and verify concurrent systems and processes, and provide operators such as synchronisation, communication, and various flavours of composition. The semantics of a process algebra is often given by means of a structural operational semantics (SOS) which associates a labelled transition system (LTS) to each term. Equivalence, and preorders, can be defined over the semantics where two terms are identified whenever no observer can notice any difference between their external behaviours. Thus equivalences and preorders can be defined in terms of a function O that represents the *set* of observations one could make while interacting with a process. For every such O we can define $p \sqsubseteq_O q$ iff $O(q) \subseteq O(p)$ and $p =_O q$ iff $O(p) = O(q)$. Varying how the environment *interacts* with a process leads to differing observations and these can be thought of as differing *testing scenarios*, and thus different preorders (i.e., refinement relations).

These are detailed in the literature, and an overview and comprehensive treatment is provided by van Glabbeek in [26,27]. As in [14] we do not consider internal or silent events here, and the relationship between differing equivalences and refinement relations is hence often given by the linear-time, branching-time spectrum given in Figure 1.

The testing scenarios described in [27] are found by modelling a process as a black box that contains an interface to the environment, via which tests are performed. Varying the interface gives different testing scenarios, a full characterisation is given in [25], for example, the interface might contain a display in which the name of the action is shown that is currently carried out by the process, buttons might also be present (one for each action) so that the observer may determine which actions are free and which are blocked, or lamps which illuminate if the process is ready to engage in that action.

We assume the usual notation for labelled transition systems (LTSs):

Definition 11 (Labelled Transition Systems (LTSs))
A labelled transition system is a tuple $(States, Act, T, s_0)$ *where* $States$ *is a non-empty set of states,* $s_0 \in States$ *is an initial state,* Act *is a set of actions, and* $T \subseteq States \times Act \times States$ *is a transition relation.* □

Every state in the LTS represents a process itself – namely the one that the original one (the initial state s_0) evolves to after following a specific path in the LTS. Specific notation needed includes the usual notation for writing transitions as $p \xrightarrow{a} q$ for $(p, a, q) \in T$ and the extension of this to traces (written $p \xrightarrow{tr} q$) and the set of initial actions of a process which is defined as:

$next(p) = \{a \in Act \mid \exists q \bullet p \xrightarrow{a} q\}$.

In the remainder of this section we detail differing preorders and show how they are embedded into our relational model. For each we give its definition, its characterisation as a testing scenario as described by van Glabbeek, its embedding into a relational model, and thereby the definition of simulation rules to characterise the preorder.

3.1 Trace preorder

3.1.1 Definition and testing scenario

Definition 12 $\sigma \in Act^*$ *is a trace of a process* p *if* $\exists q \bullet p \xrightarrow{\sigma} q$. $\mathcal{T}(p)$ *denotes the set of traces of* p*. The trace preorder is defined by* $p \sqsubseteq_{tr} q$ *iff* $\mathcal{T}(q) \subseteq \mathcal{T}(p)$. □

Testing scenario: Observations consist of a sequence of actions performed by the process in succession, that is, the interface is just a display which shows the name of the action that is currently carried out by the process, and the name remains visible in the display if deadlock occurs (unless deadlock occurs initially).

3.1.2 Relational embedding

As observed previously [14] the partial relations model records exactly trace information for the embedding with trivial finalisation described in Section 2.3. Possible traces lead to the single global value; impossible traces have no relational image.

Definition 13 (Trace embedding)
A Z data type $(State, Init, \{Op_i\}_{i \in I})$ *has the following trace embedding into the*

relational model.

$$\begin{aligned}
&\mathsf{G} == \{*\}\\
&\mathsf{State} == State\\
&\mathsf{Init} == \{Init \bullet * \mapsto \theta State'\}\\
&\mathsf{Op} == \{Op \bullet \theta State \mapsto \theta State'\}\\
&\mathsf{Fin} == \{(\theta State, *)\}
\end{aligned}$$

□

Observe also that the finalisation is a total function, thus conditions involving inclusions between finalisations will simplify to equalities. To prove the correspondence between trace preorder and data refinement we need to provide a definition of the traces of an abstract data type.

Definition 14 *The* traces *of a Z data type* $(State, Init, \{Op_i\}_{i \in I})$ *are all sequences* $\langle i_1, \ldots, i_n \rangle$ *such that*

$$\exists\, State' \bullet Init \,⨾\, Op_{i_1} \,⨾\, \ldots \,⨾\, Op_{i_n}$$

We denote the traces of an ADT A by $\mathcal{T}(A)$*.* □

Theorem 3.1 *With the trace embedding, data refinement corresponds to trace preorder. That is, when Z data types* A *and* C *are embedded as* A *and* C [3]*,*

$$\mathsf{A} \sqsubseteq_{data} \mathsf{C} \text{ iff } \mathcal{T}(C) \subseteq \mathcal{T}(A)$$

Proof From the definition of traces for Z data types and the embedding given it is obvious that for any sequence p, $(*, *) \in p_{\mathsf{A}}$ iff $p \in \mathcal{T}(A)$. Also, for any p, $p_{\mathsf{A}} = \{(*, *)\}$ or $p_{\mathsf{A}} = \varnothing$. Thus, data refinement ($p_{\mathsf{A}} \subseteq p_{\mathsf{C}}$ for all p) corresponds to trace refinement. □

From this result it can be seen that observations in the testing scenario, here a display with an action name displayed, are distributed in the relational notion of refinement. That is, although finalisations are often taken to be the 'observations', in fact, some of the observations are implicit in the program p and the relational inclusion $p_{\mathsf{C}} \subseteq p_{\mathsf{A}}$ (since finalisations only contain the information as to whether the trace was defined or not).

We can now extract the simulation rules that correspond to this notion of refinement. These are of course the rules for standard Z refinement but omitting applicability of operations, as used also e.g. in Event-B [1].

[3] This condition is left implicit in the rest of this paper.

3.1.3 Simulations

The conditions for a downward simulation in the partial relational model are (c.f. Definition 4):

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

The first and last of these are just the standard initialisation and correctness conditions, respectively. The finalisation condition in fact places no further requirements with the trace embedding. The same is true for upwards simulations. We thus have the following conditions for the trace embedding.

Definition 15 (Trace simulations in Z)
Given Z data types A and C, the relation R on $AState \wedge CState$ is a trace downward simulation *from A to C if*

$$\begin{array}{l} \forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \wedge R' \\ \forall\, i \in I \bullet \forall\, AState;\ CState;\ CState' \bullet R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i \end{array}$$

The total relation T on $AState \wedge CState$ is a trace upward simulation *from A to C if*

$$\begin{array}{l} \forall\, AState';\ CState' \bullet CInit \wedge T' \Rightarrow AInit \\ \forall\, i : I \bullet \forall\, AState';\ CState;\ CState' \bullet \\ \qquad (COp_i \wedge T') \Rightarrow (\exists\, AState \bullet T \wedge AOp_i) \end{array}$$

□

3.2 Completed trace preorder

3.2.1 Definition and testing scenario

Definition 16 *$\sigma \in Act^*$ is a completed trace of a process p if $\exists\, q \bullet p \xrightarrow{\sigma} q$ and $next(q) = \varnothing$. $\mathcal{CT}(p)$ denotes the set of completed traces of p. The completed trace preorder, $\sqsubseteq_{ctr}$, is defined by $p \sqsubseteq_{ctr} q$ iff $\mathcal{T}(q) \subseteq \mathcal{T}(p)$ and $\mathcal{CT}(q) \subseteq \mathcal{CT}(p)$.* □

Testing scenario: Observations consist of a sequence of actions performed by the process in succession, that is, the interface is just a display which shows the name of the action that is currently carried out by the process, where the display becomes empty if deadlock occurs.

3.2.2 Relational embedding

Definition 17 (Completed trace embedding)
The Z data type $(State, Init, \{Op_i\}_{i \in I})$ has the following completed trace embedding

into the relational model.

$$\begin{aligned}
&\mathsf{G} == \{*, \surd\}\\
&\mathsf{State} == State\\
&\mathsf{Init} == \{Init \bullet * \mapsto \theta State'\}\\
&\mathsf{Op} == \{Op \bullet \theta State \mapsto \theta State'\}\\
&\mathsf{Fin} == \{State \bullet \theta State \mapsto *\} \cup \{State \mid (\forall\, i : I \bullet \neg\, \mathrm{pre}\, Op_i) \bullet \theta State \mapsto \surd\}
\end{aligned}$$

□

Here the global state has been augmented with an additional element $\surd$, which denotes that the given trace is complete (i.e., no operation is applicable).

Definition 18 *The* completed traces *of a Z data type* $(State, Init, \{Op_i\}_{i\in I})$ *are all sequences* $\langle i_1, \ldots, i_n\rangle$ *such that*

$$\exists\, State' \bullet Init \mathbin{⨟} Op_{i_1} \mathbin{⨟} \ldots \mathbin{⨟} Op_{i_n} \wedge \forall\, i : I \bullet \neg(\mathrm{pre}\, Op_i)'$$

We denote the complete traces of an ADT A by $\mathcal{CT}(A)$. □

Theorem 3.2 *With the completed trace embedding, data refinement corresponds to completed trace preorder. That is,*

$$\mathsf{A} \sqsubseteq \mathsf{C} \textit{ iff } \mathcal{CT}(C) \subseteq \mathcal{CT}(A) \textit{ and } \mathcal{T}(C) \subseteq \mathcal{T}(A)$$

Proof 1. Suppose that $\mathcal{CT}(C) \subseteq \mathcal{CT}(A)$ and $\mathcal{T}(C) \subseteq \mathcal{T}(A)$. To show $A \sqsubseteq C$ we need $p_{\mathsf{C}} \subseteq p_{\mathsf{A}}$ for all programs p. Given p, if p is not a trace of C then $p_{\mathsf{C}} = \varnothing$, and thus the inclusion is trivial. Otherwise, either $(*, \surd)$ and $(*, *)$ are both in p_{C} or just $(*, *)$ is in p_{C}.

If $(*, \surd)$ is in p_{C} then p is a completed trace in C, and thus also in A. Hence $(*, \surd)$ is in p_{A}, and so is $(*, *)$. If just $(*, *)$ is in p_{C} then p is a trace which is not a completed trace in C. Since $\mathcal{T}(C) \subseteq \mathcal{T}(A)$, p is also a trace in A. Hence $(*, *)$ is in p_{A}.

2. Suppose $A \sqsubseteq C$.

Given $p \in \mathcal{CT}(C)$. Thus $(*, \surd) \in p_{\mathsf{C}} \subseteq p_{\mathsf{A}}$, and hence $p \in \mathcal{CT}(A)$. For a similar reason we also get trace inclusion. □

We can now extract the simulation rules that correspond to this notion of refinement.

3.2.3 Simulations

Given the completed trace embedding in the relational model, only the finalisation is altered from the embedding given in Section 3.1. Thus we just have to consider the effect of the finalisation requirement:

Downward simulations: $\mathsf{R} \mathbin{⨟} \mathsf{CFin} \subseteq \mathsf{AFin}$ is equivalent to

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

Upward simulations: $\mathsf{CFin} \subseteq \mathsf{T} \mathbin{;} \mathsf{AFin}$ is equivalent to

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

We thus have the following conditions for the trace embedding.

Definition 19 (Completed trace simulations in Z)
Given Z data types A and C. The relation R on AState ∧ CState is a completed trace downward simulation *from A to C if*

$$\begin{array}{l}
\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \land R' \\
\forall\, i \in I \bullet \forall\, AState;\ CState;\ CState' \bullet R \land COp_i \Rightarrow \exists\, AState' \bullet R' \land AOp_i \\
\forall\, AState;\ CState \bullet R \land \forall\, i : I \bullet \neg \operatorname{pre} COp_i \Rightarrow \forall\, i : I \bullet \neg \operatorname{pre} AOp_i
\end{array}$$

The total relation T on AState ∧ CState is a completed trace upward simulation *from A to C if*

$$\begin{array}{l}
\forall\, AState';\ CState' \bullet CInit \land T' \Rightarrow AInit \\
\forall\, i \in I \bullet \forall\, AState';\ CState;\ CState' \bullet \\
\qquad (COp_i \land T') \Rightarrow (\exists\, AState \bullet T \land AOp_i) \\
\forall\, CState \bullet \forall\, i : I \bullet \neg \operatorname{pre} COp_i \Rightarrow \exists\, AState \bullet T \land \forall\, i : I \bullet \neg \operatorname{pre} AOp_i
\end{array}$$

□

3.3 Failure preorder

3.3.1 Definition and testing scenario

The failures semantics records both the traces that a process can do, and also sets of actions which it can refuse, that is, actions which are not enabled. These are recorded as failures of a process.

Definition 20 *$(\sigma, X) \in Act^* \times \mathbb{P}(Act)$ is a failure of a process p if there is a process q such that $p \xrightarrow{\sigma} q$, and $next(q) \cap X = \varnothing$. $\mathcal{F}(p)$ denotes the set of failures of p. The failures preorder, $\sqsubseteq_f$, is defined by $p \sqsubseteq_f q$ iff $\mathcal{F}(q) \subseteq \mathcal{F}(p)$.* □

Testing scenario: The machine for testing failures has, in addition to the interface of the completed trace machine, a switch for each action in Act. One can then observe which actions are blocked. If the process reaches a state where all actions are blocked, then this can be observed by an empty display. Observations are thus the failures of a process.

3.3.2 Relational embedding

This was covered in detail in [5,14,4], although we used an embedding into the totalised relational model there. Lemma 3 in [4] suggested this was not necessary: $\perp$ appears as a possible outcome iff somewhere along the trace the next action of the trace could be refused. Thus, below we give a simpler embedding into the partial relations model.

Definition 21 (Failures embedding)
A Z data type $(State, Init, \{Op_i\}_{i \in I})$ in the refusals interpretation is embedded in the relational model as follows.

$$\begin{aligned}
&\mathsf{G} == \mathbb{P}\, I \\
&\mathsf{State} == State \\
&\mathsf{Init} == \{Init;\ E : \mathbb{P}\, I \bullet E \mapsto \theta State'\} \\
&\mathsf{Op} == \{Op \bullet \theta State \mapsto \theta State'\} \\
&\mathsf{Fin} == \{State;\ E : \mathbb{P}\, I \mid (\forall\, i \in E \bullet \neg \operatorname{pre} Op_i) \bullet \theta State \mapsto E\}
\end{aligned}$$

□

In the relational embedding failures are pairs (tr, X), where tr is a trace, and there exists states $(\mathsf{State}, \mathsf{State}') \in tr$ (with State being initial) such that $\forall\, i : X \bullet \mathsf{State}' \not\in \operatorname{dom} \mathsf{Op}_i$.

Theorem 3.3 *With the failures embedding, data refinement corresponds to the failures preorder. That is,*

$$A \sqsubseteq C \textit{ iff } \mathcal{F}(C) \subseteq \mathcal{F}(A)$$

The proof of this is an adaptation of that given in [14]. □

3.3.3 Simulations

Given the failures embedding the changes to the simulation conditions are as follows (these are derived in [14] - remember we have no input/output at this stage):

Downward simulations: $\mathsf{R} \,⨾\, \mathsf{CFin} \subseteq \mathsf{AFin}$ is equivalent to

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

Upward simulations: $\mathsf{CFin} \subseteq \mathsf{T} \,⨾\, \mathsf{AFin}$ is equivalent to

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

3.4 Failure trace preorder

3.4.1 Definition and testing scenario

The failure trace semantics considers refusal sets not only at the end of a trace, but also between each action in a trace.

Definition 22 *$\sigma \in (Act \cup \mathbb{P}\, Act)^*$ is a failure trace of a process p if $\sigma = X_1 a_1 X_2 a_2 \ldots X_n a_n X_{n+1}$ where $a_1 a_2 \ldots a_n$ is a trace of p and each $(a_1 \ldots a_i, X_{i+1})$ is a failure of p. $\mathcal{FT}(p)$ denotes the set of failure traces of p. The failures traces preorder, $\sqsubseteq_{ftr}$, is defined by $p \sqsubseteq_{ftr} q$ iff $\mathcal{FT}(q) \subseteq \mathcal{FT}(p)$.* □

Testing scenario: The display in the machine for testing failures traces is the same as that for failures. However, it does not halt if the process cannot proceed, rather it idles until the observer allows one of the actions the process is ready to perform. The observations are traces with idle periods in between, and for each idle period the set of actions that are not blocked by the observer.

It has been argued [19,20] that this is a better notion for testing than simply observing failures of a process, and is appropriate when one can detect that a process refuses an action, and if this is the case, one has the ability to try another action.

3.4.2 Relational embedding

Definition 23 (Failure trace embedding)
A Z data type $(State, Init, \{Op_i\}_{i \in I})$ *in the failure trace interpretation is embedded in the relational model as follows.*

$$
\begin{array}{l}
\mathsf{G} == \mathrm{seq}\,\mathbb{P}\, I \\
\mathsf{State} == \mathrm{seq}\,\mathbb{P}\, I \times State \\
\mathsf{Init} == \{Init;\ fs : \mathrm{seq}\,\mathbb{P}\, I \bullet fs \mapsto (\langle\,\rangle, \theta State')\} \\
\mathsf{Op} == \{Op;\ fs : \mathrm{seq}\,\mathbb{P}\, I;\ E : \mathbb{P}\, I \mid (\forall\, i : E \bullet \neg\,\mathrm{pre}\; Op_i) \bullet \\
\qquad\qquad (fs, \theta State) \mapsto (fs \frown \langle E \rangle, \theta State')\} \\
\mathsf{Fin} == \{State;\ fs : \mathrm{seq}\,\mathbb{P}\, I \mid (\forall\, i : E \bullet \neg\,\mathrm{pre}\; Op_i) \bullet (fs, \theta State) \mapsto fs\}
\end{array}
$$

□

In the relational embedding failures traces are the obvious generalisation of failures.

Theorem 3.4 *With the failure traces embedding, data refinement corresponds to the failure traces preorder. That is,*

$$A \sqsubseteq C \ \textit{iff}\ \mathcal{FT}(C) \subseteq \mathcal{FT}(A)$$

□

3.4.3 Simulations

In the failure trace embedding, both the correctness and finalisation conditions are potentially amended due to the record of failures at each operation step.

Downward simulations: Here, in fact, the finalisation condition is subsumed by the correctness, and $\mathsf{R} \mathbin{⨾} \mathsf{COp}_i \subseteq \mathsf{AOp}_i \mathbin{⨾} \mathsf{R}$ expands to the following

$$
\begin{array}{l}
\forall\, AState;\ CState;\ CState' \bullet \forall\, E \bullet \\
\qquad R \wedge COp_i \wedge Fcond(E, \theta CState') \ \Rightarrow \\
\qquad \exists\, AState' \bullet R' \wedge AOp_i \wedge Fcond(E, \theta AState')
\end{array}
$$

where $Fcond(E, s) == \forall\, i : E \bullet \neg\,\exists\, Op_i \bullet s = \theta State.$

Upward simulations: Similarly, $\mathsf{COp}_i \mathbin{⨾} \mathsf{T} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AOp}_i$ expands to

$$
\begin{array}{l}
\forall\, AState';\ CState;\ CState' \bullet \forall\, E \bullet \\
\qquad (COp_i \wedge T' \wedge Fcond(E, \theta CState')) \ \Rightarrow \\
\qquad \exists\, AState \bullet T \wedge AOp_i \wedge Fcond(E, \theta AState')
\end{array}
$$

3.5 *Ready preorder*

3.5.1 *Definition and testing scenario*

An alternative to the various failures semantics and preorders are semantics based upon acceptance sets, that is, semantics recording the actions a process is willing to engage in, rather than its refusal sets.

Definition 24 *$(\sigma, X) \in Act^* \times \mathbb{P}(Act)$ is a ready pair of a process p if there is a process q such that $p \stackrel{\sigma}{\longrightarrow} q$, and $next(q) = X$. $\mathcal{R}(p)$ denotes the set of ready pairs of p. The readiness preorder, $\sqsubseteq_r$, is defined by $p \sqsubseteq_r q$ iff $\mathcal{R}(q) \subseteq \mathcal{R}(p)$.* □

Testing scenario: In the readiness semantics observations consist of a trace of a process or a trace together with the set of actions which the observation could have been extended with (if the observer had wished). The machine corresponding to this has a light for each action, which illuminate if the process is ready to engage with that action.

3.5.2 *Relational embedding*

The relational embedding is mostly as in the failures embedding, using the same global state and initialisation. However, rather than finalising to *any* set of events that may be refused, we now finalise to *the* set of events that must be accepted.

Definition 25 (Readiness embedding)
A Z data type $(State, Init, \{Op_i\}_{i \in I})$ in the readiness interpretation is embedded in the relational model as follows.

$$
\begin{array}{l}
\mathsf{G} == \mathbb{P}\, I \\
\mathsf{State} == State \\
\mathsf{Init} == \{Init;\ E : \mathbb{P}\, I \bullet E \mapsto \theta State'\} \\
\mathsf{Op} == \{Op \bullet \theta State \mapsto \theta State'\} \\
\mathsf{Fin} == \{State \bullet \theta State \mapsto \{i \in I \bullet \mathrm{pre}\ Op_i\}\}
\end{array}
$$

□

Theorem 3.5 *With the readiness embedding, data refinement corresponds to the readiness preorder. That is,*

$$A \sqsubseteq C \text{ iff } \mathcal{R}(C) \subseteq \mathcal{R}(A)$$

Proof See [14]. □

3.5.3 *Simulations*

Given the readiness embedding the changes to the simulation conditions are as follows (these are derived in [14]):

Downward simulations: $\mathsf{R} \fatsemi \mathsf{CFin} \subseteq \mathsf{AFin}$ is equivalent to

$$\forall AState;\ CState;\ i : I \bullet R \Rightarrow \mathrm{pre}\ COp_i \Leftrightarrow \mathrm{pre}\ AOp_i$$

Upward simulations: $\mathsf{CFin} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AFin}$ is equivalent to

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

3.6 Ready trace preorder

The ready trace semantics is similar to the failure trace semantics, except acceptance sets replace failures in the traces observed. The relational embedding is the same as that for the failure trace, except with the substitution of an appropriate acceptance set. Similar derivation can be made for the simulation rules.

3.7 Extension and conformance

3.7.1 Definition and testing scenario

Although not considered in [27], a number of alternative preorders for process algebras have been suggested motivated by testing and the need for test generation. Specifically, in the context of testing from LOTOS specifications [6] these have included *extension* and *conformance* [10]. To define these formally we need the following notation which defines refusals sets after a particular trace.

Definition 26 (Refusals after a trace)
Let p be an LTS, σ a trace of p, and $X \subseteq Act$. Then p **after** *σ* **ref** *X iff*

$$\exists\, q \bullet p \xrightarrow{\sigma} q \textit{ and } X \cap next(q) = \varnothing$$

□

Testing scenario: Three definitions of refinement can be given on the basis of the idea behind Definition 26. These were motivated in [10,11] by considering that there might be a number of different notions of implementation:

- implementation as a real/physical system
- implementation as a (deterministic) reduction of a given specification
- implementation as a (conforming) extension of a given specification
- implementation as a refinement of a given specification

These are formalised [9] by, respectively, conformance, reduction, extension and testing equivalence. Reduction (also called the testing preorder [12]) in our context (of no divergence) is identical to the failures preorder. Testing equivalence is the equivalence induced by that preorder.

Conformance has the following characteristics: if $p \sqsubseteq_{conf} q$ then q deadlocks less often than p when in any environment whose traces are limited to those of q. Thus conformance restricts the quantification (of traces one must check refusals about) to be over the abstract specification (and this restriction gives rise to efficient test generation algorithms).

The *extension* preorder can be defined as conformance together with the additional property that traces can be extended. Thus, if $p \sqsubseteq_{ext} q$ then q has at least the same traces as p, but in an environment whose traces are limited to those of p, it deadlocks less often. The equivalence induced by extension is the same as that by reduction (that is, testing equivalence). Leduc [21] documents the relationship between these relations in some detail. They can be defined as follows.

Definition 27 (Reduction, conformance, and extension)
Let p, q be LTSs. Then

$$p \sqsubseteq_{red} q \textit{ iff } \forall \sigma : Act^*;\ X \subseteq Act \bullet q \textbf{ after } \sigma \textbf{ ref } X \textit{ implies } p \textbf{ after } \sigma \textbf{ ref } X$$

$$p \sqsubseteq_{conf} q \textit{ iff } \forall \sigma : \mathcal{T}(p);\ X \subseteq Act \bullet q \textbf{ after } \sigma \textbf{ ref } X \textit{ implies } p \textbf{ after } \sigma \textbf{ ref } X$$

$$\begin{array}{l} p \sqsubseteq_{ext} q \textit{ iff} \\ \quad \mathcal{T}(p) \subseteq \mathcal{T}(q) \textit{ and} \\ \quad \forall \sigma : \mathcal{T}(p);\ X \subseteq Act \bullet q \textbf{ after } \sigma \textbf{ ref } X \textit{ implies } p \textbf{ after } \sigma \textbf{ ref } X \end{array}$$

□

3.7.2 Relational embedding

The relational embedding we use to model extension is, in fact, a totalisation over the space of partial relations, and is the standard *non-blocking* model (e.g., as discussed in [15]), that is, we use the same construction to record the effect of extension in a blocking model as we did to record failures in a non-blocking model.

Definition 28 (Extension embedding)
A Z data type $(State, Init, \{Op_i\}_{i \in I})$ in the extension interpretation is embedded in the relational model as follows.

$$\begin{array}{l} \mathsf{G} == \mathbb{P}\, I \cup \{\bot\} \\ \mathsf{State} == State \cup \{\bot\} \\ \mathsf{Init} == \{Init;\ E : \mathbb{P}\, I \bullet E \mapsto \theta State'\} \\ \mathsf{Op} == \mathsf{OpB} \cup \{\mathsf{x}, \mathsf{y} : \mathsf{State} \mid \mathsf{x} \notin \mathrm{dom}\, \mathsf{OpB} \bullet (\mathsf{x}, \mathsf{y})\} \\ \textit{where } \mathsf{OpB} == \{Op \bullet \theta State \mapsto \theta State'\} \\ \mathsf{Fin} == \{State;\ E : \mathbb{P}\, I \mid (\forall\, i : E \bullet \neg\, \mathrm{pre}\ Op_i) \bullet \theta State \mapsto E\} \cup \{(\bot, \bot)\} \end{array}$$

□

Theorem 3.6 *With the extension embedding, data refinement corresponds to the extension preorder. That is,*

$$\begin{array}{l} A \sqsubseteq C \textit{ iff} \\ \quad \mathcal{T}(A) \subseteq \mathcal{T}(C) \textit{ and} \\ \quad \forall \sigma \in \mathcal{T}(A) \cap \mathcal{T}(C);\ X \subseteq Act \bullet C \textbf{ after } \sigma \textbf{ ref } X \textit{ implies } A \textbf{ after } \sigma \textbf{ ref } X \end{array}$$

Proof 1. Suppose that $\mathcal{T}(A) \subseteq \mathcal{T}(C)$ and $\forall \sigma \in \mathcal{T}(A) \cap \mathcal{T}(C);\ X \subseteq Act \bullet C$ **after** σ **ref** X implies A **after** σ **ref** X.

Given $(g, E) \in p_{\mathsf{C}}$ then $tr \in \mathcal{T}(C)$. Either $tr \in \mathcal{T}(A)$ in which case refusal inclusion gives us $(g, E) \in p_{\mathsf{A}}$ or $tr \notin \mathcal{T}(A)$ in which case the non-blocking totalisation gives $(g, E), (g, \perp) \in p_{\mathsf{A}}$.

2. Suppose $A \sqsubseteq C$. Then trace inclusion can be proved by induction over the length of the trace, and refusals subsetting follows as a consequence of using the non-blocking totalisation. $\square$

Whilst we have found an embedding such that data refinement induces extension, this is not possible for conformance. This is because conformance is *not* a preorder (see any of the references given above), but data refinement *is* a preorder. Thus no combinations of embeddings as a data refinement theory will produce an embedding equivalent to it. Intuitively this is because the quantification over programs, and the subsetting of program behaviour contains at its very heart an embedding of trace inclusion. However, conformance makes no requirement about trace inclusion, from either the concrete to the abstract or vice versa, and is just concerned with refusals, and hence cannot be modeled this way.

3.7.3 Simulations

The use of the non-blocking totalisation for modelling extension means we can extract simulation conditions by reference to above results. They are thus the following.

Definition 29 (Extension downward simulation in Z)
Given Z data types A and C. The relation R on AState ∧ CState is a extension downward simulation *from A to C if*

$$\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \wedge R'$$
$$\forall\, i : I \bullet \forall\, AState;\ CState \bullet \operatorname{pre} AOp_i \wedge R \Rightarrow \operatorname{pre} COp_i$$
$$\forall\, i : I \bullet \forall\, AState;\ CState;\ CState' \bullet \operatorname{pre} AOp_i \wedge R \wedge COp_i \Rightarrow \exists\, AState' \bullet R' \wedge AOp_i$$

$\square$

Definition 30 (Extension upward simulation in Z)
Given Z data types A C. The total relation T on AState ∧ CState is an extension upward simulation *from A to C if*

$$\forall\, AState';\ CState' \bullet CInit \wedge T' \Rightarrow AInit$$
$$\forall\, CState \bullet \exists\, AState \bullet \forall\, i : I \bullet T \wedge (\operatorname{pre} AOp_i \Rightarrow \operatorname{pre} COp_i)$$
$$\forall\, i : I \bullet \forall\, AState';\ CState;\ CState' \bullet$$
$$(COp_i \wedge T') \Rightarrow (\exists\, AState \bullet T \wedge (\operatorname{pre} AOp_i \Rightarrow AOp_i))$$

$\square$

Note the symmetry: failure-divergence for the non-blocking model is extension for the blocking model.

4 Conclusions

In this paper we have derived simulations for relational embeddings of a number of refinement preorders found in process algebras.

Although downward and upward simulations (Definitions 4 and 5) are complete, their totalised versions are not. However, complete simulations can be given for each semantics, e.g. the failures semantics simulations are known to be complete. A separate completeness proof for simulations is needed in each embedding, this waits for an extended version of this paper.

References

[1] Abrial, J.-R., D. Cansell and D. Méry, *Refinement and reachability in event$_b$*, in: H. Treharne, S. King, M. C. Henson and S. A. Schneider, editors, *ZB*, Lecture Notes in Computer Science **3455** (2005), pp. 222–241.

[2] Bergstra, J. A., A. Ponse and S. A. Smolka, editors, "Handbook of Process Algebra," Elsevier Science Inc., New York, NY, USA, 2001.

[3] Boiten, E. and J. Derrick, *Incompleteness of relational simulations in the blocking paradigm* (2008), submitted for publication.

[4] Boiten, E., J. Derrick and G. Schellhorn, *Relational concurrent refinement II: Internal operations and outputs*, Formal Aspects of Computing Accepted for publication. URL http://www.cs.kent.ac.uk/pubs/2007/2633

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

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

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

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

[9] Brinksma, E., *A theory for the derivation of tests*, in: S. Aggarwal and K. Sabnani, editors, *Protocol Specification, Testing and Verification, VIII* (1988), pp. 63–74.

[10] Brinksma, E. and G. Scollo, *Formal notions of implementation and conformance in LOTOS*, Technical Report INF-86-13, Dept of Informatics, Twente University of Technology (1986).

[11] Brinksma, E., G. Scollo and C. Steenbergen, *Process specification, their implementation and their tests*, in: B. Sarikaya and G. v. Bochmann, editors, *Protocol Specification, Testing and Verification, VI* (1986), pp. 349–360.

[12] de Nicola, R., *Extensional equivalences for transition systems*, Acta Informatica **24** (1987), pp. 211–237.

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

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

[15] Derrick, J. and E. A. Boiten, "Refinement in Z and Object-Z," Springer-Verlag, 2001.

[16] He Jifeng and C. A. R. Hoare, *Prespecification and data refinement*, in: *Data Refinement in a Categorical Setting*, Technical Monograph, number PRG-90, Oxford University Computing Laboratory, 1990 .

[17] He Jifeng, C. A. R. Hoare and J. W. Sanders, *Data refinement refined*, in: B. Robinet and R. Wilhelm, editors, *Proc. ESOP 86*, Lecture Notes in Computer Science **213** (1986), pp. 187–196.

[18] Hoare, C. A. R., "Communicating Sequential Processes," Prentice Hall, 1985.

[19] Langerak, R., *A testing theory for LOTOS using deadlock detection*, in: *Protocol Specification Testing and Verification IX* (1989), pp. 87–98.

[20] Langerak, R., "Transformations and Semantics for LOTOS," Ph.D. thesis, University of Twente, The Netherlands (1992).

[21] Leduc, G., "On the Role of Implementation Relations in the Design of Distributed Systems using LOTOS," Ph.D. thesis, University of Liège, Liège, Belgium (1991).

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

[23] Reeves, S. and D. Streader, *Data refinement and singleton failures refinement are not equivalent*, Formal Aspects of Computing Accepted for publication.

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

[25] van Glabbeek, R., *The linear time – branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract)*, in: E. Best, editor, Proceedings *CONCUR'93*, 4^{th} International Conference on *Concurrency Theory*, Hildesheim, Germany, August 1993, Lecture Notes in Computer Science **715** (1993), pp. 66–81.

[26] van Glabbeek, R. J., *The linear time - branching time spectrum*, in: J. C. M. Baeten and J. W. Klop, editors, *CONCUR 90*, Lecture Notes in Computer Science **458** (1990), pp. 278–297.

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

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