# Incompleteness of Relational Simulations in the Blocking Paradigm

Eerke Boiten[1] , John Derrick[2]
[1] School of Computing, University of Kent
Canterbury, Kent, CT2 7NF, UK E.A.Boiten@kent.ac.uk
[2] Department of Computer Science, University of Sheffield,
Sheffield, S1 4DP, UK J.Derrick@dcs.shef.ac.uk

July 8, 2010

**Abstract**

Refinement is the notion of development between formal specifications. For specifications given in a relational formalism, downward and upward simulations are the standard method to verify that a refinement holds, their usefulness based upon their soundness and joint completeness. This is known to be true for total relational specifications and has been claimed to hold for partial relational specifications in both the *non-blocking* and *blocking* interpretations.

In this paper we show that downward and upward simulations in the blocking interpretation, where domains are "guards", are *not* jointly complete. This contradicts earlier claims in the literature. We illustrate this with an example (based on one recently constructed by Reeves and Streader) and then construct a proof to show why joint completeness fails in general.
**Keywords:** relational refinement, downward and upward simulations, joint completeness, failures refinement.

## 1 Introduction

In relational state-based specification formalisms, there are at least three fundamentally different approaches to *partial* relations, which are described below. Informally, the central question is often phrased as: "what happens" when an operation is applied outside its domain? However, it is really an issue of refinement, i.e., of acceptable substitution of behaviour: when an operation's specification is partial, how does this constrain its implementations? This is normally viewed in terms of "blocking": if a state is outside an operation's domain, should the data type deadlock in that state when its environment insists on performing that operation?

Consider the data type with initial state 0 in Figure 1(a). Informally, we call the outcomes of a trace (a sequence of actions) the set of states we may end up in by following the actions one by one, starting from an initial state. Thus, the outcomes for $b$ are 1 and 2, for $bb$ just 2, and for $bbb$ there are none.

The first approach is the "contract" or "non-blocking" approach to abstract data types, traditionally used for a specification language such as Z [23]. Here, the domain of a relation specifies the area within which the operation is guaranteed to deliver a well-defined result. The domain is thus a "precondition", and we view the operation's effect outside its domain as "anything may happen". This interpretation is shown in Figure 1(c). Here it is possible to do two $b$'s, the first ending in state 2, and the second leading to all possible outcomes. If we use (as is standard) $\perp$ to denote the error state, we thus find that trace $bb$ has all states including $\perp$ as an outcome. Under a refinement,

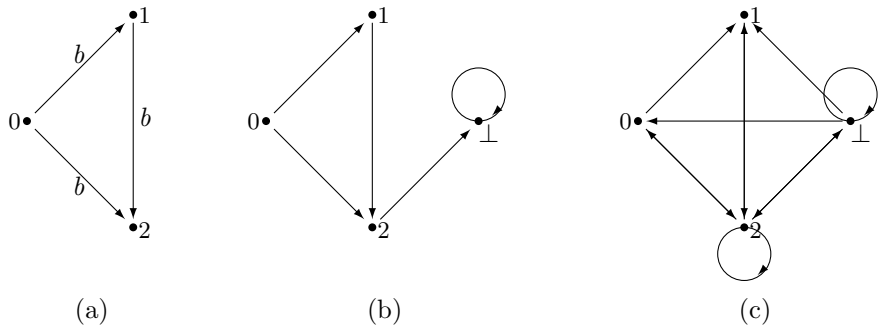


Figure 1: A data type with initial state 0 and partial operation $b$ (not applicable in state 2) in (a). Different interpretations of partiality: in (b), "blocking" adds an error state $\perp$ with transition from state 2 to $\perp$; in (c), "non-blocking" also adds transitions from 2 and $\perp$ to all states. Operation labels $b$ have been omitted on all transitions in (b) and (c).

where we can reduce non-determinism, we can reduce a result which was a non-deterministic choice of all possibilities including errors to any specific value, thus an operation's precondition can can be weakened in refinement.

The second approach is the "behavioural" or "blocking" model, illustrated in Figure 1(b). Here, the domain of an operation is interpreted as a "guard": it is impossible to apply the operation outside the guard. This can be encoded by representing the outcome of applying an operation outside its domain by a special error value $\perp$. Then, $bb$ has outcomes 2 and $\perp$, and $bbb$ has only $\perp$. Under a refinement this behaviour must be preserved, and consequently the guard cannot be weakened in such a blocking approach.

The third approach, related to the second, consists in not encoding partiality at all. This results in an interpretation of non-determinism which is angelic as far as deadlock is concerned, observing certain deadlock but not possible deadlock. In the example, it means staying with Figure 1(a), where the outcomes for $bb$ hide that the second $b$ was impossible in state 2. That is, for any given trace, we can observe if executing it will *always* block on one of its actions (e.g., $bbb$), but not the situation where this may or may not happen depending on the data type's resolution of non-determinism (e.g., $bb$). In this notion of refinement, guards may even be strengthened, preserving "nothing bad happens" – leading to the notion of trace refinement.

Refinement is the notion of development between formal specifications. Various theories of refinement for *relational* formalisms have been developed – see De Roever and Engelhardt [6] for a historical overview and analysis. Extensive bibliographic information can also be found in [16, 7]. At the heart of all theories of refinement is a notion that in a development one should be allowed to substitute one specification (or indeed implementation) for another provided the behaviour is consistent - that is, one cannot *observe* the difference. This idea allows one to reduce non-determinism as discussed in the context of the small example above. Clearly also from the example above, non-blocking refinement has a slightly different characterisation to blocking refinement, due to their difference in handling the domain of a partial relation.

For specifications given in a relational formalism, downward and upward simulations are the standard (tractable) method to verify that a refinement holds, their usefulness based upon their soundness and joint completeness. Such a theory of simulations for total relations was described by Hoare, He and Sanders, initially in [12] and later in more detail in [11]. Simulations relate state spaces of two data types in a way that ensures refinement holds, through a step-by-step comparison of each operation.

For total relations (i.e., a specification consisting of relations all of which are total), and for

the non-blocking refinement of partial relations, downward and upward simulations are sound and jointly complete. In this paper we show that the simulation rules in the blocking approach are *incomplete*, contradicting claims to the contrary in the literature. The problem arises for the following reason.

For a specification $C$ which is a refinement of $A$, the joint completeness result constructs an intermediate specification $B$ and a simulation from $A$ to $B$ and another from $B$ to $C$. For total relations clearly $B$ is also a total relational specification. For a partial relational specification the proof is adapted so that $C$ and $A$ are *totalised*, that is turned into total relations via the addition of the error state as in the example above. One then finds, since they are now all total relations, a specification $B$ as before. However, one must now also check that this $B$ could have arisen from an underlying partial specification. In the non-blocking approach one can - which is why the joint completeness result still holds. However, in the blocking approach one cannot. Providing a counterexample, understanding the underlying reasons for failure, and then exhibiting a general proof, is thus the contribution we make in this paper.

In the next section, we give the well-known simulation rules of Hoare, He and Sanders, their essential properties, and in the following section the various sets of rules for partial relations deriving from them. The subsequent section contains our main result: that the simulation rules in the blocking approach are incomplete, contradicting claims to the contrary in the literature. Then, in Section 5 we discuss this result in the context of a subtly different set of simulation rules, viz. those for failures refinement. This explains the origin of the result, and provides an elegant indirect proof. We conclude in Section 6.

## 2 Data Types and Relational Simulations

First, we give the standard theory of refinement and simulations of total data types, as introduced in [12, 11], using the definitions and notations of [7].

Relational data types are centred around a hidden local state space $\mathsf{S}$. However, their semantics are defined in terms of observations on a visible "global" state space $\mathsf{G}$. These observations are induced by *programs* which are 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 a relation on the global state: an initialisation, followed by operations on the local state, followed by a finalisation.

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

A *program* over a data type $\mathsf{D}$ 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 $\mathsf{p}_\mathsf{D}$, is the relational composition $\mathsf{Init} \fatsemi \mathsf{p} \fatsemi \mathsf{Fin}$.

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 $\mathsf{p}_\mathsf{C} \subseteq \mathsf{p}_\mathsf{A}$. □

Thus, refinement is defined in terms of relational inclusion of all programs. For total data types clearly each program is defined to be a non-empty relation. Thus refinement requires that the concrete observations are consistent with what could have been produced by the corresponding abstract program. For total data types, every program produces an observation, so in that case with a one-element global state refinement will always hold. For partial data types, it first appears that a refinement may be trivially satisfied by choosing empty concrete operations. However, this

actually depends on what observations are made by the finalisation, and many finalisations do not allow such a trivial behaviour.

To make the verification of refinement tractable, the standard methodology is to use simulations, which relate state spaces of two data types in a way that ensures refinement holds, through a step-by-step comparison of each operation. There are two varieties: downward and upward simulations.

**Definition 2 (Downward simulation)**
Consider total data types $\mathsf{A} = (\mathsf{AS}, \mathsf{AI}, \{\mathsf{AOp}_i\}_{i \in I}, \mathsf{AF})$ and $\mathsf{C} = (\mathsf{CS}, \mathsf{CI}, \{\mathsf{COp}_i\}_{i \in I}, \mathsf{CF})$. A relation $\mathsf{R}$ between $\mathsf{AS}$ and $\mathsf{CS}$ is a *downward* simulation between $\mathsf{A}$ and $\mathsf{C}$ if it satisfies the three conditions:

$$\mathsf{CI} \subseteq \mathsf{AI} \fatsemi \mathsf{R}$$
$$\mathsf{R} \fatsemi \mathsf{CF} \subseteq \mathsf{AF}$$
$$\forall\, i : I \bullet \mathsf{R} \fatsemi \mathsf{COp}_i \subseteq \mathsf{AOp}_i \fatsemi \mathsf{R}$$

□

**Definition 3 (Upward simulation)**
A relation $\mathsf{T}$ between $\mathsf{CS}$ and $\mathsf{AS}$ is an *upward* simulation between total data types $\mathsf{A}$ and $\mathsf{C}$ as above if it satisfies the three conditions:

$$\mathsf{CI} \fatsemi \mathsf{T} \subseteq \mathsf{AI}$$
$$\mathsf{CF} \subseteq \mathsf{T} \fatsemi \mathsf{AF}$$
$$\forall\, i : I \bullet \mathsf{COp}_i \fatsemi \mathsf{T} \subseteq \mathsf{T} \fatsemi \mathsf{AOp}_i$$

□

The relevance of simulations is explained by the following.

**Theorem 1** Upward and downward simulation are *sound* and *jointly complete* for data refinement, i.e., the existence of either kind of simulation ensures that refinement holds between the respective data types, and any data refinement can be proved using a sequence of upward and downward simulations only.

We give the essential aspects of the proof only. Soundness is proved by induction over the program.

Joint completeness is proved by the construction of an intermediate data type $\mathsf{B}$ such that an upward simulation exists from $\mathsf{A}$ to $\mathsf{B}$, and a downward simulation from $\mathsf{B}$ to $\mathsf{C}$. A useful lemma is that all refinements of data types with a deterministic initialisation and operations can be proved using downward simulation only. The intermediate data type $\mathsf{B}$ is constructed as a deterministic normal form of $\mathsf{A}$: either using the familiar powerset construction (as from NDFAs to DFAs), or as a "canonical data type" where every state represents a different trace. This normal form construction is indeed an upward simulation, which postpones all non-deterministic choice to the finalisation. (As the normal form is equivalent to its original, an alternative proof is by exhibiting a downward simulation between both data types' normal forms.)

Hoare, He and Sanders presented this theory initially [12] for total data types only, but in subsequent papers such as [11] removed the totality restriction without emphasizing that fact. In the form given above, the simulation rules are thus also sound and jointly complete for partial operations. This, then, provides a first theory of simulations for partial data types, e.g. for verifying trace refinement, and they are used in this way by Lynch and Vaandrager [16] and others.

# 3 Simulation rules from totalised relations

As indicated above, the two main theories for partial relations are the blocking and non-blocking ones which are derived from the total relations theory. Indeed, a specification language such as Z gains its theory and methodology of refinement from using a blocking or non-blocking approach depending on the meaning given to the area outside an operation's domain (i.e., are we specifying guards or preconditions).

A possible derivation of the simulation rules for a theory of partial relations consists of four steps: extending the state spaces; embedding operations (etc.) in the enhanced state spaces by "totalisation"; applying the simulation theory to the totalised data types; and then expressing the resulting conditions in terms of the original state and operations.

The global and local state spaces are extended by adding a new value $\bot$ to the state space which represents "erroneous" behaviour. We let $\mathsf{S}_\bot = \mathsf{S} \cup \{\bot\}$ for some $\bot \notin \mathsf{S}$. Augmenting the state space with such a value allows one to make each operation total in a way that encodes how an operation is defined outside its domain. Thus totalisation is defined as follows.

**Definition 4 (Totalisation)**
For a partial relation $\mathsf{Op}$ on $\mathsf{S}$, its *totalisation* is a total relation on $\mathsf{S}_\bot$, defined in the non-blocking model by

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

and in the blocking model by

$$\widehat{\mathsf{Op}}^{\mathsf{b}} == \mathsf{Op} \cup \{\mathsf{x} : \mathsf{S}_\bot \mid \mathsf{x} \notin \mathrm{dom}\, \mathsf{Op} \bullet (\mathsf{x}, \bot)\}$$

Finalisation is already assumed to be total on $\mathsf{S}$, but it is totalised on $\mathsf{S}_\bot$ by adding $(\bot, \bot)$, thus making erroneous behaviour observable. □

For an example, see Figure 1, where the single operation $b$ of the partial data type in (a) is given a blocking totalisation in (b) and a non-blocking one in (c).

The complete derivation of the non-blocking theory was (to our knowledge) first published by Woodcock and Davies [23]. The blocking theory with its embedding was described by Bolton et al [5], without details of the derivation. Our monograph [7] contains full details of both; Deutsch and Henson [9] have recently investigated the design choices for totalisations in the blocking theory. We have studied variants with multiple level embeddings in [1, 2].

The simulation rules for partial operations are derived by applying the simulation rules to the totalised relations, and then eliminating $\bot$ from the resulting conditions. Thus, the rules inherit soundness from soundness of the standard rules; however, completeness does not transfer immediately.

To understand why completeness might be an issue, consider the schematic diagram given in Figure 2. The circles represent partial data types. These are then totalised in the manner described above, leading to an embedding of a partial data type as a total data type. This embedding is depicted as the circle being part of the large oval - the space of total specifications. To attempt to prove completeness, take $A$ and $C$ and consider their totalisation - the standard theory constructs an intermediate *total* data type, here called $A_{nf}$, and simulations from it to the totalised $A$ and $C$.

The key question now is whether there are simulations from the partial $A$ and $C$ to some intermediate $B$ - which is also a *partial* data type. For this to be the case, $A_{nf}$ needs to be derivable as a totalised partial data type - in terms of the picture it must lie within the circle, not just the oval. This is not automatically the case, and is the source of the failure of completeness for the blocking simulation rules.

As they play no further rôle in this paper, we do not give the non-blocking simulation rules here. The only relevant aspect is the following.

**Theorem 2** The non-blocking simulation rules are sound and jointly complete for partial data types in the non-blocking interpretation.

This theorem is, to our knowledge, not stated and proved anywhere for the standard non-blocking simulations. However, De Roever and Engelhardt [6, pp. 187-188] prove a closely related theorem.

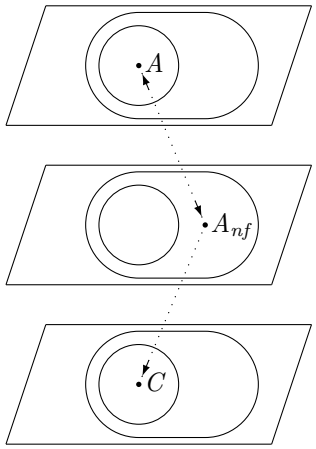


Figure 2: The completeness proof of the underlying theory (represented by the oval) does not automatically transfer to completenes for an embedded theory (the circle indicates the image of the embedding); arrows indicate simulations, $A_{nf}$ is the "normal form" of data type $A$, and lies outside the image of the embedding.

Their non-blocking rules are derived from the *partial* relations simulations of He and Hoare, so their formalism observes both (certain) blocking and (possible) divergence using possibly partial relations on $\mathsf{S}_\perp$; see [1] for a discussion. Their theorem does not imply the theorem we are looking for (as they operate in a larger domain of relations), but its proof carries over successfully. The proof inherits the construction of the proof of Theorem 1 and recognises the additional proof obligation: that the intermediate datatype is in the image of the embedding[1], avoiding the situation in Figure 2.

In the *blocking* case, the totalisation resembles a well-known construction from automata theory. From an NDFA which does not have transitions from all states for all actions, a "total" NDFA is created by adding a "sink" state, and adding all the missing transitions with the sink state as their target (plus loops from the sink to itself for every action), exactly as in Figure 1(b). Additionally, the observations (finalisations) from the states are defined to be the normal observations for all original states, and a special error value for the sink. In the particular case of a partial data type with a single-element global state $\{*\}$, which is extended to $\{*, \perp\}$, all original local states represent "success" $*$, and the new state $\perp$ represents failure. However, crucially, in the definition of recognition of languages by NDFAs, the non-determinism is interpreted *angelically*, i.e., asking whether a string *can* lead to success – *not* distinguishing between the outcome sets $\{\perp, *\}$ and $\{*\}$ as the relational semantics does. Thus, in Figure 1(b), in the NDFA interpretation the string $bb$ is simply accepted, whereas its relational view returns both $*$ and $\perp$.

The simulation rules that are derived from the Hoare, He and Sanders rules through this construction are as follows.

**Definition 5 (Downward simulation for partial relations (blocking))**
Consider data types $\mathsf{A}$ and $\mathsf{C}$ as in Definition 2 where the operations may be partial. A *downward*

[1]Their embedding is not explicit, instead they characterise relations in its image axiomatically.

simulation is a relation $\mathsf{R}$ from $\mathsf{AS}$ to $\mathsf{CS}$ satisfying[2]

$$\mathsf{CI} \subseteq \mathsf{AI} \fatsemi \mathsf{R}$$
$$\mathsf{R} \fatsemi \mathsf{CF} \subseteq \mathsf{AF}$$
$$\forall\, i : I \bullet \mathrm{ran}(\mathrm{dom}\, \mathsf{AOp}_i \lhd \mathsf{R}) \subseteq \mathrm{dom}\, \mathsf{COp}_i$$
$$\forall\, i : I \bullet \mathsf{R} \fatsemi \mathsf{COp}_i \subseteq \mathsf{AOp}_i \fatsemi \mathsf{R}$$

The conditions are refered to as *initialisation*, *finalisation*, *applicability* and *correctness*. □

**Definition 6 (Upward simulation for partial relations (blocking))**
Assume data types $\mathsf{A}$ and $\mathsf{C}$ as above. An *upward* simulation is a relation $\mathsf{T}$ from $\mathsf{CS}$ to $\mathsf{AS}$ satisfying

$$\mathsf{CI} \fatsemi \mathsf{T} \subseteq \mathsf{AI}$$
$$\mathsf{CF} \subseteq \mathsf{T} \fatsemi \mathsf{AF}$$
$$\forall\, i : I;\ \mathsf{c} : \mathsf{CS} \bullet \exists\, \mathsf{a} : \mathsf{AS} \bullet (\mathsf{c}, \mathsf{a}) \in \mathsf{T} \land (\mathsf{a} \in \mathrm{dom}\, \mathsf{AOp}_i \Rightarrow \mathsf{c} \in \mathrm{dom}\, \mathsf{COp}_i)$$
$$\forall\, i : I \bullet \mathsf{COp}_i \fatsemi \mathsf{T} \subseteq \mathsf{T} \fatsemi \mathsf{AOp}_i$$

The conditions are again refered to as *initialisation*, *finalisation*, *applicability* and *correctness*. □

These simulation rules were first considered systematically by Bolton et al [5]. However, they had already been used (postulated, rather than derived) within the Z community before this [22] and in particular they were presented as the simulation rules for Object-Z (however, see the discussion in Sections 5 and 6).

# 4 Blocking simulations are not jointly complete

Bolton, Davies and Woodcock [5] state a theorem that the above simulation rules are sound and complete. Their proof of completeness is on the same basis as the completeness proofs described above: by exhibiting an intermediate data type, with an upward simulation from the abstract to the intermediate, and a downward simulation from intermediate to concrete.

However, unlike the proof of Theorem 2 outlined above, the proof in [5] omits the proof obligation that the intermediate data type is in the image of the embedding used. This proof obligation turns out not to hold, i.e., a situation as in Figure 2 arises.

Informally, if we start with a single-element global state $\{*\}$, the embedding enhances this to $\{*, \bot\}$. The embedding of the finalisation links the extra local state $\bot$ to $\bot$, and all "normal" states to $*$. The intermediate data type which postpones all non-determinism to the finalisation would finalise some states to $*$, some states to $\bot$, and some to both of these. The last kind of state does not arise in data types constructed by the embedding, and thus the intermediate data type will in general not be an image of the embedding.

This invalidates the proof in [5] – but not (yet) the theorem itself. We postpone a formal proof that the theorem itself does not hold until Section 5. The crucial element in this proof is the example in Figure 3.

In this figure two partial data types $A$ and $C$ are specified, each with a single operation $b$. They are totalised in the blocking model, where transitions added by totalisation are indicated with dotted arrows but we have omitted the $b$ transition from $\bot$ to itself. Blocking relational refinement holds from $A$ to $C$, however, this cannot be proved using simulations.

**Lemma 1** For the data types in Figure 3, $\mathsf{C}$ is a relational refinement of $\mathsf{A}$ in the blocking model.

[2] $P \lhd R$ is the relation $R$ constrained to the domain $P$, i.e., $\{(x, y) : R \mid x \in P\}$.

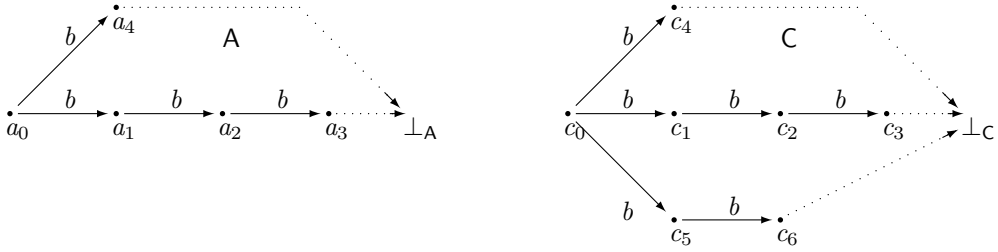


Figure 3: Relational (blocking) refinement - later on we will also use the fact that $C$ is not a (singleton) failures refinement of $A$.

**Proof** The relation between traces and final observations for $\mathsf{A}$ is $\{(\epsilon, *), (b, *), (bb, *), (bbb, *)\} \cup bb^+ \times \{\bot\}$ and it is the same for $\mathsf{C}$. □

The next lemma is straightforward.

**Lemma 2** *There is no blocking downward simulation from* $\mathsf{A}$ *to* $\mathsf{C}$.

**Proof** Construction of a relation $\mathsf{R}$ satisfying the downward simulation conditions is not possible. From the initialisation condition it follows that $(a_0, c_0) \in \mathsf{R}$. Then from correctness $\mathsf{R} \fatsemi \mathsf{b_C} \subseteq \mathsf{b_A} \fatsemi \mathsf{R}$ it follows that $(a_0, c_5) \in \mathsf{b_A} \fatsemi \mathsf{R}$. This implies that either $(a_4, c_5) \in \mathsf{R}$ or $(a_1, c_5) \in \mathsf{R}$. The former fails on correctness; the latter implies $(a_1, c_6) \in \mathsf{b_A} \fatsemi \mathsf{R}$, hence $(a_2, c_6) \in \mathsf{R}$, which fails on applicability. □

However, a more surprising lemma is the following.

**Lemma 3** $\mathsf{C}$ is not a blocking upward simulation of $\mathsf{A}$ (i.e., following Definition 6) although an upward simulation (according to Definition 3) exists between their embeddings.

**Proof** We construct an upward simulation $\mathsf{T}$ between the embeddings, showing that the conditions of Definition 6 cannot be satisfied. Note that the embeddings are indicated in Figure 3 including all arrows and dotted arrows as the definition of the single operation $b$.

From totality of finalisations it follows that upward simulations are total on the concrete state (in either definition), so we need to find one or more abstract states matching $c_6$. The correctness condition works "backward", so we can only link $c_6$ to abstract states that allow two preceding $b$ operations. We perform a case analysis on those states:

$(c_6, a_3) \in \mathsf{T}$**:** From correctness it follows that also $(c_5, a_2) \in \mathsf{T}$. Then by correctness also $(c_0, a_1) \in \mathsf{T}$. This violates the initialisation condition, as $a_1 \notin \operatorname{ran} \mathsf{AI}$. Thus, this case cannot lead to a solution.

$(c_6, a_2) \in \mathsf{T}$**:** This is not allowed by the applicability condition (which is unique to Definition 6), which states that if $b$ is impossible in $c_6$ and $c_6$ is linked to $a_2$, then $b$ should also be impossible in $a_2$. Thus, this case is also excluded.

Given that there are no other states that $c_6$ could be linked to, the conditions of Definition 6 cannot be satisfied.

However, the second case is *not* ruled out by Definition 3 and we find that the following upward simulation holds between the embeddings:

$$\mathsf{T} = \{(c_i, a_i)\}_{i=0..4} \cup \{(c_5, a_1), (c_5, a_4), (c_6, a_2), (c_6, \bot_{\mathsf{A}}), (\bot_{\mathsf{C}}, \bot_{\mathsf{A}})\}$$

The mapping of $c_6$ corresponds to postponement of non-determinism in refinement, which is the situation where upward simulation tends to be necessary: the state $c_6$ represents both $a_2$ and "$\bot$ after $a_4$". □

This proves the following preliminary incompleteness lemma.

**Lemma 4** In the blocking model, partial data types exist such that they are *not* related by upward simulation (in the sense of Definition 6), whereas their corresponding embeddings *are* related by upward simulation (in the sense of Definition 3). □

This still does not prove joint incompleteness of blocking upward and downward simulation: for situations like this, a different sequence of intermediate data types might exist to circumvent the problem. Given the small size of the example, one could prove the absence of such a sequence by exhaustively considering all relational refinements of $\mathsf{A}$ with at most $n$ branches for some credible value of $n$ (3? 6?) – however, a more elegant proof is found below via the connection to failures refinement.

# 5 Simulations for Refusals

In this section, we prove the incompleteness of blocking simulations by interpreting the counterexample using relational failure semantics.

Going back to the late 1980s, the group around the Oxford PRG not only investigated semantics and refinement for relational data types and Z, but also for CSP [13, 19]. In particular, *failures refinement* was defined in both settings – in the relational model, upward and downward simulations for it were characterised and shown to be sound and complete by Josephs [15], He [14] and Woodcock and Morgan [24].

The results in this paper rely solely on results for this particular interpretation, which have been proved by Josephs [15] and others. We do not formally define failures semantics and refinement here – informally, a failure of a process $P$ is a pair $(t, E)$ such that $P$ can perform trace $t$ and afterwards refuse all events in the set $E$. *Singleton* failures semantics considers only sets $E$ of at most one element. Failures refinement of processes is defined as set inclusion on their failures, and similarly for singleton failures refinement. The crucial definition is the following.

**Definition 7 (Failure simulations [15])** For data types $\mathsf{A}$ and $\mathsf{C}$ above, simulation relations are defined as follows.

- Downward simulations are characterised by Definition 5.
- Upward simulations are characterised by Definition 6, with the additional condition

$$\forall \mathsf{c} : \mathsf{CS} \bullet \exists \mathsf{a} : \mathsf{AS} \bullet (\mathsf{c}, \mathsf{a}) \in \mathsf{T} \land \forall i : I \bullet \mathsf{a} \in \operatorname{dom} \mathsf{AOp}_i \Rightarrow \mathsf{c} \in \operatorname{dom} \mathsf{COp}_i \tag{1}$$

□

The following theorem is proved by [15, 24, 14].

**Theorem 3** The upward and downward simulation rules of Definition 7 are sound and jointly complete for failures refinement. □

Although often confused, refinement induced by the blocking simulation rules is distinct from failures refinement, as shown by Bolton et al [4, 3] who illustrated the distinction with a number of small examples. It is intuitively unsurprising that all examples in Bolton et al's papers illustrating the difference use data types with at least two operations. The following lemma appears to demonstrate the necessity of this.

**Lemma 5** When the index set $I$ is a singleton set, the blocking simulation rules (Definitions 5 and 6) and failure simulation rules (Definition 7) coincide.

**Proof** Condition (1) and applicability in Definition 6 are equivalent when the quantification over $i : I$ is trivial. All other conditions are identical. □

Our final result is a corollary of these results. First, we need to revisit Figure 3. So far we have shown that $C$ is a refinement of $A$ in the blocking model. However, although it is a relational blocking refinement, it is *not* a singleton failures refinement. To see this note that the singleton failure arising in $\mathsf{C}$ but not in $\mathsf{A}$ is $(bb, \{b\})$ in state $c_6$: $C$ can do $bb$ and then refuse another $b$, but $A$ cannot.

We can now glue together these results as applied to this example to derive our contradiction.

**Theorem 4** The blocking simulation rules of Definitions 5 and 6 are not jointly complete.

**Proof** Lemma 1 shows that blocking refinement holds between the data types in Figure 3. We assume completeness of blocking simulations, and derive a contradiction. Completeness means that a sequence of blocking upward and downward simulation steps exists to prove the refinement. By Lemma 5 each of these steps is also a failures simulation. By Theorem 3 (soundness of failure simulations) this establishes a failures refinement between the data types. However, singleton failures refinement does not hold, and thus *a fortiori* failures refinement does not hold either. Consequently, such a sequence of simulations cannot exist and the blocking simulation rules are incomplete. □

# 6 Concluding Comments

Overall, blocking data refinement may be viewed as somewhat problematic. Its relation with intuitive semantic models is less than straightforward [4, 3, 18]. Moreover, the simulation rules that come "naturally" with it are incomplete. We might do better to adopt failures refinement overall: its semantics is well understood, and it coincides with data refinement on forward simulation.

We are reminded that the strategy of deriving rules through embeddings, though attractive in principle, carries a risk: it guarantees soundness, but does not preserve completeness. The completeness proof of the simulation rules for failures refinement, e.g., bears little relation to their reconstruction through embedding.

In the absence of internal actions and hiding, the relational simulation rules for failures refinement are very similar to the blocking simulations defined in Definitions 5 and 6. For a survey of the literature on comparing relational and process algebraic refinement relations see [2]. Although the essence of the correspondence between relational data types and process algebra is intuitive – both can be viewed as transition systems – there are many subtle issues.

For example, due to the great similarity between condition (1) in Definition 7 and the applicability condition in Definition 6 – further obscured if the quantification over $i$ in (1) is expressed in terms of *next* sets as it is in [15] – the two notions of refinement they represent have frequently been confused. Woodcock and Morgan in 1990 [24] hinted at a difference, by stating that their *forward* simulation rules are identical to the sequential ones (our emphasis). However, we are not aware of any paper published between then and 2002 that mentions both the "blocking rules" and "failures refinement" but does *not* mistakenly identify them – see e.g. [21, 10, 20, 5].

Bolton et al [4, 3] finally brought the difference to the fore again. They also pointed out that the failures simulation rules were the correct ones to use for the Object-Z histories semantics. Complementary to their analysis identifying the relational blocking rules with singleton failures semantics (now shown to be incorrect by Reeves and Streader [18]), we showed [8] that using finalisations which observe failures in the relational model leads to the failure simulation rules.

The example constructed by Reeves and Streader [18] to exhibit the difference between relational refinement and singleton failure refinement is illuminating. The difference between their example and the one given in Figure 3 is that theirs concerns the trace $abc$ and its prefixes rather than $bbb$ – either would have worked to make their point, but the change is necessary for the proof given in Section 5. These two systems are equivalent in the relational semantics, but the interesting direction of refinement is (as suggested by the naming) from $A$ to $C$. In that direction, singleton failures refinement does not hold but blocking relational refinement does. The only way out of this is to use a different encoding which records exactly at which point blocking occurs – for example by effectively introducing states $\bot_n$ for all $n : \mathbb{N}$. A more detailed discussion of this is given in [17].

The results in this paper are the first examples of lack of completeness due to the embedding of the intermediate data type. A related issue is soundness, where for data types involving unbounded non-determinism soundness of upward simulations fails, a result which goes back 20 years. However, this issue arises only when programs include loops or recursion, whereas the relational theory used here (the standard one for Z and related formalisms) only considers straight line programs.


## Acknowledgements

We would like to thank Steve Reeves and David Streader for coming up with a crucial counterexample and comments on this paper, and the anonymous reviewer of [2] who pointed us at their result. We also thank Wim Hesselink and the reviewers for their comments which have clarified the contents and presentation of this paper.


# References


[1] 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.

[2] E.A. Boiten, J. Derrick, and G. Schellhorn. Relational concurrent refinement part II: Internal operations and outputs. *Formal Aspects of Computing*, 21(1-2):65–102, 2009.

[3] C. Bolton and J. Davies. A comparison of refinement orderings and their associated simulation rules. In J. Derrick, E.A. Boiten, J.C.P. Woodcock, and J. von Wright, editors, *REFINE 02: The BCS FACS Refinement Workshop*, volume 70(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, July 2002.

[4] 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.

[5] 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.

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

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

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

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

[10] C. Fischer and H. Wehrheim. Failure-divergence semantics as a formal basis for an object-oriented integrated formal method. *Bulletin of the EATCS (European Association of Theoretical Computer Science)*, 71:92 – 101, 2000.

[11] 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, November 1990.

[12] 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.

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

[14] He Jifeng. Process simulation and refinement. *Formal Aspects of Computing*, 1(3):229–241, 1989.

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

[16] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations: I. Untimed systems. *Inf. Comput.*, 121(2):214–233, 1995.

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

[18] S. Reeves and D. Streader. Data refinement and singleton failures refinement are not equivalent. *Formal Aspects of Computing*, 20(3):295–301, 2008.

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

[20] A. Simpson, J. Davies, and J.C.P. Woodcock. Security management via Z and CSP. In J. Grundy, M. Schwenke, and T. Vickers, editors, *International Refinement Workshop & Formal Methods Pacific '98*, Discrete Mathematics and Theoretical Computer Science, Canberra, September 1998. Springer-Verlag.

[21] G. Smith and J. Derrick. Refinement and verification of concurrent systems specified in Object-Z and CSP. In M.G. Hinchey and S. Liu, editors, *First International Conference on Formal Engineering Methods (ICFEM'97)*, pages 293–302, Hiroshima, Japan, November 1997. IEEE Computer Society Press.

[22] B. Strulo. How firing conditions help inheritance. In J.P. Bowen and M.G. Hinchey, editors, *ZUM'95: The Z Formal Specification Notation*, volume 967 of *Lecture Notes in Computer Science*, pages 264–275, Limerick, September 1995. Springer-Verlag.

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

[24] 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.