Getting to the Bottom of Relational Refinement: Relations and Correctness, Partial and Total

Eerke Boiten[1] and Willem-Paul de Roever[2]

[1] University of Kent, Canterbury, UK
[2] Christian Albrechts University, Kiel, Germany

Abstract. Relational theories of refinement using simulations exist based on total and on partial relations, modelling partial and total correctness. In order to model partialities such as non-termination and deadlock, domains of relations are often extended with, or assumed to include, a "bottom" value $\perp$. This paper explores a number of such theories, each of which provides sound and jointly complete notions of simulations, concentrating on the rôle of $\perp$.

1 Introduction

This paper surveys some of the "dark corners" of relational refinement theory. In particular, it concentrates on the rôle of fictitious elements $\perp$ that are added to the domain and range of the relations involved, in order to model various partialities (such as deadlock and possible non-termination) within theories of both total and partial[1] relations. Many of the pieces of the puzzle described here are well-established, but we did not find a paper giving the full picture. Some of the relevant questions, a few of which are addressed here, are:

- How should one model: deadlock; non-termination; both?
- What does $\perp$ represent?
- What is the relevance of partial vs. total relations in representing partial and total correctness?
- What is the problem with unbounded non-determinism, and why does it only appear in certain contexts?

We begin by setting out the basic theory of refinement and simulations for relational abstract data types with *total* operations in Section 2. In Section 3 we discuss the generalisation of this theory which allows *partial* operations. An alternative interpretation of partial operations is given in Section 4. Section 5 describes a combination of the previous two interpretations; the final section discusses an even richer model where a single $\perp$ value is not enough.

[1] Note that we use "partial relations" (operations) to refer explicitly to relations that do not need to be total, i.e., relations in general; this differs from the usage for *functions* where "function" without further qualifiers refers to a total function.

2 Total Relations, Sequences and Simulations

The theory of data refinement presented in this section is a fragment of that set out by He, Hoare and Sanders in ESOP'86 [8]. Further closely related publications by the same authors, of which [9] is the most recent, contain subtle generalisations and improvements to this theory, some of which will be mentioned later. See also [2] for further analysis and references. The presentation of the theory here is based on that in [3].

Programs represent computations on a *global state* G; they are represented as binary relations. We consider a fixed G. Global computations are achieved in terms of elementary operations on a local state. This local state forms part of an *abstract data type*, which also defines in which local states computations may start (initialisation), and how "final" local states relate to global states (finalisation).

Definition 1 (Data type). *A* data type *is a tuple* $(\mathsf{L}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i \in I}, \mathsf{Fin})$*. The operations* $\{\mathsf{Op}_i\}$*, indexed by* $i \in I$*, are* total *relations on the set* L*;* Init *is a total relation from* G *to* L*;* Fin *is a total relation from* L *to* G*.* □

We fix a set of operations ("alphabet") I in the rest of this paper.

Programs are constructed from the basic operations of a data type, using a given set of program *constructors*. For the moment, we restrict this to sequential composition, so programs will be finite sequences over I. (This is the principal restriction in comparison to [8]). Programs must be interpreted in a data type, using its basic operations and the meanings of the program constructors. The resulting relation over the local state is then pre-composed with the initialisation and post-composed with the finalisation relation, all in all defining a computation on the global state. The standard semantics for sequences is sequential composition, so for example, the meaning of the program $\langle i_1, i_2 \rangle$ in the data type $(\mathsf{L}, \mathsf{Init}, \{\mathsf{Op}_i\}_{i \in I}, \mathsf{Fin})$ is $\mathsf{Init} \mathbin{⨾} \mathsf{Op}_{i_1} \mathbin{⨾} \mathsf{Op}_{i_2} \mathbin{⨾} \mathsf{Fin}$. In general, the interpretation (semantics) of a program p over I in a data type D is denoted by $[\![p]\!]_{\mathsf{D}}$. Refinement between data types is defined as compatible behaviour (modelled by relational inclusion, i.e., reduction of non-determinism) for all programs:

Definition 2 (Data refinement). *For data types* A *and* C*,* C refines A *iff for each program* p *over* I*,* $[\![\mathsf{p}]\!]_{\mathsf{C}} \subseteq [\![\mathsf{p}]\!]_{\mathsf{A}}$*.* □

The standard method of making the verification of refinement tractable is to use *simulations*, where a step-by-step comparison is made of each operation in the two data types. To do so it is necessary to link the concrete and abstract states. This leads to the situation described in Figure 1, with the requirement that the triangles and squares semi-commute. An inclusion of the concrete path in the abstract path can be proved by certain combinations of semi-commutativity (i.e., inclusion of a path involving the concrete type into one involving the abstract type) of the subdiagrams. The two most useful ways to do this are known as *downward* (forward, or L- [2]) simulation, and *upward* (backward, or L^{-1}-) simulation, defined as follows.

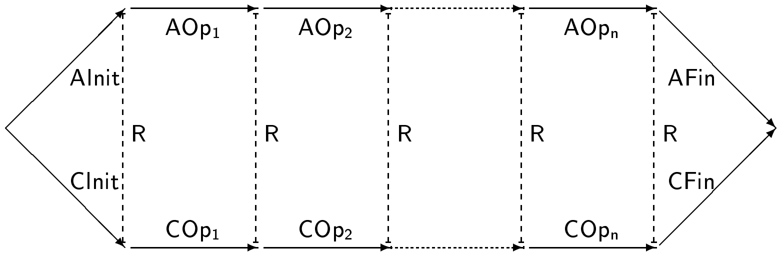

Fig. 1. Data refinement with simulations.

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

$$\mathsf{CInit} \subseteq \mathsf{AInit} \mathbin{⨾} \mathsf{R} \tag{1}$$

$$\mathsf{R} \mathbin{⨾} \mathsf{CFin} \subseteq \mathsf{AFin} \tag{2}$$

$$\forall\, i : I \bullet \mathsf{R} \mathbin{⨾} \mathsf{COp}_i \subseteq \mathsf{AOp}_i \mathbin{⨾} \mathsf{R} \tag{3}$$

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*.* □

Definition 4 (Upward simulation). *For data types* A *and* C *as above, an* upward *simulation is a relation* T *from* CS *to* AS *such that*

$$\mathsf{CInit} \mathbin{⨾} \mathsf{T} \subseteq \mathsf{AInit} \tag{4}$$

$$\mathsf{CFin} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AFin} \tag{5}$$

$$\forall\, i : I \bullet \mathsf{COp}_i \mathbin{⨾} \mathsf{T} \subseteq \mathsf{T} \mathbin{⨾} \mathsf{AOp}_i \tag{6}$$

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*.* □

In practice, many refinements can be proved by just backward simulation; in particular this is the case when the abstract type is "canonical", i.e., all operations are functions. Upward simulation is necessary when a refinement involves the postponement of a non-deterministic choice.

When programs are finite sequences, we have the following theorem without further restrictions:

Theorem 1 (Soundness and completeness of simulations). *Upward and downward simulation are both sound, and jointly complete with respect to the definition of data refinement.*

Soundness is proved by structural induction over the program. One might view simulations as a "canned induction" proof for refinement: rather than always using induction for proving inclusion between all corresponding programs (Definition 2), we reduce the base cases and induction steps to conditions not requiring induction, once and for all.

Joint completeness uses a powerset construction (an upward simulation) to define a canonical variant of the abstract type; then it invokes the completeness result for downward simulations with canonical abstract types.

In order to classify this model as a "total correctness" or "partial correctness" model, a source of "undefinedness" (divergence, non-termination, deadlock, ...) is required, which could be found in the operations (initialisation, finalisation), in the program constructors, or both[2]. Sequential composition does not introduce undefinedness, of course. Total operations by themselves do not lead to undefinedness, thus here is little sense in calling this either a "total correctness" or "partial correctness" model. However, one might interpret one or more values ("$\perp$") in the range of the operations as "undefinedness", and when considering recursion or while loops, one is forced to do so to model possible non-termination. Such a model is considered in more detail in Section 4.

This model abstracts a number of more concrete situations. It serves as the semantic basis of refinement in Z [14, 3], with inputs and outputs in the global state, and the local state containing those plus a "state schema". Imperative programming is modelled by the global state containing all "global" ("normal" [2]) variables, and the local state containing local variables as well.

The full theory of [8] contains two major extensions to the one presented here. First, as basic building blocks also "conditions" (partial identity relations) are included. Through composition with conditions, total operations may become partial programs. Secondly, they add recursion as intersections of directed families, implicitly indicating they are assuming a Smyth cpo (and citing [13]). Soundness of upward simulation, as a consequence, requires a restriction to finitary simulation relations. The assumption of $\perp$ in the domains and ranges of operations is left implicit in [8], and made explicitly in [9, 2].

3 Partial Relations for Partial Correctness

The restricted theory of the previous section can be relaxed to apply equally well to *partial* relations, with simulations and soundness and completeness as before. This can be seen, for example, in the proof of the completeness theorem in [9], which does not rely on totality of the operations. (The paper itself is ambiguous concerning the restriction to totality.)

If no distinguished value is included to represent undefinedness, this set-up allows the observation of certain undefinedness: when the initial global state is outside the domain of the program. However, possible undefinedness cannot be observed: if some paths through the program do not lead to a defined state,

[2] Much of the complication in refinement theories derives from the possibility to move undefinedness and non-determinism *between* operations and constructors.

but due to non-determinism there are other paths which do, only the latter are visible. Thus, this framework is a partial correctness one.

De Roever and Engelhardt [2, chapter 4] give a proof of soundness and joint completeness of simulations in this setting. Their proof shows that there is no need for finitariness restrictions on various relations, even in the presence of recursion.

Derrick and Boiten [4] recently used this result to define simulations to verify *readiness* refinement [11] for Z. In defining this readiness semantics, it is essential *not* to observe refusals, and thus the traditional approach for Z simulations (see Section 4), which observes possible blocking, is too strong for this.

A notable weakness of this partial correctness theory, following directly from its notion of refinement, is the fact that a data type whose operations and finalisation are all empty is guaranteed to be a refinement of *any* other one. However, in many application contexts, the finalisations, which provide the "observation" of a data type's effect, represent *implicit* effects such as producing the output, projection onto external variables, etc, and will thus not be empty. In general, totality of initialisation represents the fact that the data type can "always be started", and totality of finalisation means that its final result is always observable. For example, in the context of a readiness semantics, finalisations observe which operations are enabled, which ensures that refinement preserves enabling [4].

However, when observation of refusals is not a problem, powerful finalisations are not the only method for defining refinement of partial relations. The next section describes an alternative approach, going back to the total relations of the previous section.

4 Total Correctness from Partial Relations

Given a theory of simulations for total relations, one can also construct a theory of simulations for *partial* relations by embedding partial relations into total relations of a particular shape, and then extracting requirements on the partial relations from the simulation conditions of Section 2. This construction underlies the refinement theory for Z [14, 3]. A crucial decision in this context is whether an operation, when invoked outside its domain, should lead to "arbitrary" behaviour, or to deadlock. The former implies that the domain is a "precondition", which is the standard interpretation for Z [14]. The latter implies that it is a "guard", which is the standard interpretation in Object-Z [12] and in combinations of Z or Object-Z with process algebra [7, 3]. (Some formalisms, e.g., B, allow explicit specifications of either or both of these, but this requires a model, e.g., predicate transformers, which can also express infeasible specifications.) Partial relations on state space S are then interpreted as total relations on $\mathsf{S}_\perp = \mathsf{S} \cup \{\perp\}$ for some value $\perp \notin \mathsf{S}$. The "bottom" value $\perp$ represents either blocking (deadlock – for guards) or a general state of "undefinedness", often interpreted as non-termination (for preconditions). This needs to be represented explicitly in order to ensure that it is preserved by relational composition.

Definition 5 (Totalisation). *For a partial relation* Op *on* S*, its totalisation is a total relation* $\widehat{\mathsf{Op}}$ *on* $\mathsf{S}_\perp$*, defined in the "preconditions" (non-blocking) approach by*

$$\widehat{\mathsf{Op}} == \mathsf{Op} \cup \{(\mathsf{x}, \mathsf{y}) : \mathsf{S}_\perp \times \mathsf{S}_\perp \mid \mathsf{x} \notin \mathit{dom}\, \mathsf{Op}\}$$

or in the "guards" (blocking) approach by

$$\widehat{\mathsf{Op}} == \mathsf{Op} \cup \{(\mathsf{x}, \perp) \mid \mathsf{x} \notin \mathit{dom}\, \mathsf{Op}\} \qquad \square$$

Similar modifications are made to initialisation and finalisation – in particular, finalisation needs to connect $\perp$ to $\perp$.

In particular, because $\perp \notin \mathrm{dom}\, \mathsf{Op}$, the relations considered all contain $(\perp, \perp)$ in the guards approach, and in the preconditions approach we have

$$\mathsf{K}^\mathsf{S} =_{def} \{\perp\} \times \mathsf{S} \quad \subseteq \ \widehat{\mathsf{Op}} \tag{7}$$

However, there is no unique smallest relation in either case: relations resulting from totalisation are total, but the intersection of all such relations is $\{(\perp, \perp)\}$ for guards, and K^S for preconditions, neither of which is total. There is a unique largest relation in the preconditions approach, which is

$$\mathsf{U}^\mathsf{S} =_{def} \mathsf{S}_\perp \times \mathsf{S}_\perp$$

In the preconditions approach, we also have that

$$\widehat{\mathsf{Op}} \operatorname{\S} \mathsf{K}^\mathsf{S} \subseteq \widehat{\mathsf{Op}} \tag{8}$$

This represents that, if $\widehat{\mathsf{Op}}$ may lead from some initial state to $\perp$ as a possible after state, it may also lead from there to every other possible after state. This implies that there is no distinction between possible arbitrariness and certain arbitrariness. On the other hand, the guards approach distinguishes between certain deadlock (from initial state s, $\widehat{\mathsf{Op}}$ only leads to $\perp$) and possible deadlock (from initial state s, $\widehat{\mathsf{Op}}$ may lead to $\perp$). By taking relational inclusion of programs as refinement, this implies that the blocking approach allows removal of possible deadlock, but not of certain deadlock; the preconditions approach allows removal of possible or certain non-termination.

Upward and downward simulation rules for partial relations can be derived by applying the rules in Section 2 to both variants of totalised relations, see [3] for the details. This leads to the rules in Section 2 with extra conditions on the domains of the operations: preconditions may be widened, reducing "arbitrariness"; or guards must be preserved, preserving certain deadlock.

In particular, the extra condition for downward simulation of partial operations in the preconditions approach is

$$\forall\, i : I;\ c : \mathsf{C};\ a : \mathsf{A} \bullet a \in \mathrm{dom}\, \mathsf{AOp}_i \wedge (a, c) \in \mathsf{R} \Rightarrow c \in \mathrm{dom}\, \mathsf{COp}_i$$

In [2, chapter 12] it is proved how the refinement conditions for VDM, including the above "domain condition", are sound with respect to downward simulation in a total correctness setting. Identifying $\perp$ with non-termination, the derivation in [3] additionally proves *completeness* of the VDM rules with respect to downward simulation (within the other restrictions imposed by VDM).

Further investigation of the choices of totalisation, and corresponding modifications to the simulation relations used, has been done by Deutsch and Henson [5].

Overall, it does not matter for these models how exactly $\perp$ is interpreted; whenever $\perp$ appears as the only possible outcome for a given before-state, refinement via relational inclusion means that the "partiality" represented by $\perp$ needs to be preserved; where $\perp$ appears as one of several possible outcomes, this means that the "partiality" represented by $\perp$ may be removed in a refinement step. Thus, in the models considered, possible non-termination (indistinguishable from certain non-termination) may be removed, as may possible deadlock; certain deadlock, however, may not.

5 Partial Relations for Total Correctness

De Roever and Engelhardt [2, chapters 8-9] present a relational model for total correctness, with simulations for verifying refinements. Considering only the homogeneous fragment of that theory, the relations considered are exactly all the relations satisfying two properties we obtained for $\widehat{\mathsf{Op}}$ above: (7) and (8).

However, they do not include the requirement that relations should be total. As a consequence, the resulting model is a generalisation of both the partial relations model of partial correctness (interpreting the absence of a final state as certain *deadlock*), and the precondition-totalisation of the previous section. Because only *certain* deadlock is observed, only non-termination is visible when both deadlock and non-termination are possible. This means that every such "enhanced" relation can be characterised by a relation R on S and a subset A of S, such that A contains the domain of R. The relation R represents all the behaviour where termination is guaranteed; the set A represents the area where deadlock is not certain. The original relation can be obtained by "totalising" R within A, and leaving it empty "outside" A. (Various other, often subtly different, approaches describing programs as a pair of a relation and a set exist, e.g. [1, 6, 7].)

In order to allow a notion of programs that also allows recursion, analogous to [9], a cpo on "enhanced" relations is defined, with U^{S} as the bottom element, and intersections of chains as their lubs. This ensures that upward simulations can be proved sound under finitariness restrictions as usual; as in the other models, downward simulation is sound and they are jointly complete.

Although this model still identifies certain and possible non-termination, it does not allow *possible* deadlock to be identified. In an imaginary observation of a non-deterministic system this may appear to be a problem; however, in terms of relational refinement it means that a possible deadlock may not be refined to a certain deadlock. Thus, a possible deadlock (like possible termination) may be a design error, but removal of non-determinism by refinement may eradicate it.

6 Two Bottoms

The totalisation model for "guards" presents an alternative approach, where it *is* actually possible to detect possible deadlock, and also to refine possible deadlock to certain deadlock. (The equally undesirable effect of refining possible non-termination to certain non-termination is implicit in the models we considered so far, as the two are already identified.) In order to do so, we might extend the total correctness model to include an additional value **fail**. In order to also cover recursion, a new cpo needs to be defined on the extended states, and preservation of deadlock needs to be ensured. The latter implies that $(\mathbf{fail}, \mathbf{fail})$ should be included in every relation. In order to detect and refine possible deadlock, every state outside the "guard" should be linked explicitly to **fail** only.

An alternative approach is that taken in Hoare and He's "Unifying Theories" [10], where deadlock and termination are not represented by fictitious elements added to relations, but rather by implicit boolean variables ok and $wait$, where ok' represents termination and $wait'$ is deadlock. (So, rather than extending the state by a sum, they extend it as a product.) This allows for a richer theory, allowing all combinations of certain and possible deadlock and non-termination, and a notion of after-states even when deadlock or non-termination has occurred; however, a larger number of "healthiness conditions" become necessary to exclude degenerate situations. For example, after-states may not be observable after a deadlock; deadlock and non-termination may not both be observed at the same time, etc.

References

1. R. Berghammer and H. Zierer. Relation algebraic semantics of deterministic and non-deterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.
2. W.-P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*, volume 47 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.
3. J. Derrick and E.A. Boiten. *Refinement in Z and Object-Z: Foundations and Advanced Applications*. FACIT. Springer Verlag, May 2001.
4. J. Derrick and E.A. Boiten. Relational concurrent refinement. Submitted for publication, 2003.
5. M. Deutsch and M.C. Henson. An analysis of forward simulation data refinement. In D. Bert, J.P. Bowen, S. King, and M. Walden, editors, *ZB 2003. Formal Specification and Development in Z and B*, volume 2651 of *Lecture Notes in Computer Science*, pages 148–167. Springer-Verlag, 2003.
6. H. Doornbos. A relational model of programs without the restriction to Egli-Milner constructs. In E.-R. Olderog, editor, *PROCOMET '94*, pages 357–376. IFIP, 1994.
7. C. Fischer. CSP-OZ - a combination of CSP and Object-Z. In H. Bowman and J. Derrick, editors, *Second IFIP International conference on Formal Methods for Open Object-based Distributed Systems*, pages 423–438. Chapman & Hall, July 1997.
8. 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.

9. He Jifeng and C.A.R. Hoare. Prespecification and data refinement. In *Data Refinement in a Categorical Setting*, number PRG-90 in Technical Monograph. Oxford University Computing Laboratory, November 1990.
10. C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall, 1998.
11. E.-R. Olderog and C. A. R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.
12. G. Smith. *The Object-Z Specification Language*. Kluwer Academic Publishers, 2000.
13. M.B. Smyth. Effectively given domains. *Theoretical Computer Science*, 5:257–274, 1977.
14. J.C.P. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.