Working Paper Series
ISSN 1177-777X

A robust semantics hides fewer errors

Steve Reeves and David Streader

Working Paper: 03/2009
June 10, 2009

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

A robust semantics hides fewer errors

Steve Reeves and David Streader
Department of Computer Science, University of Waikato, Hamilton, New Zealand
{stevr,dstr}@cs.waikato.ac.nz

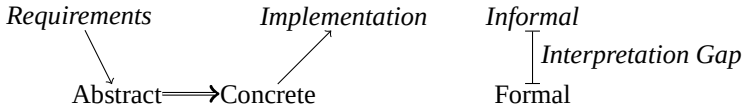
June 10, 2009

Abstract

In this paper we explore how formal models are interpreted and to what degree meaning is captured in the formal semantics and to what degree it remains in the informal interpretation of the semantics. By applying a robust approach to the definition of refinement and semantics, favoured by the event-based community, to state-based theory we are able to move some aspects from the informal interpretation into the formal semantics.

1 Introduction

As engineers proud of our reputations and our subject we want aeroplanes to fly and banks to be trustworthy. As engineers we want real systems to function in the way we have specified them and for there to be no unpleasant surprises. But with increasing complexity we, like members of all other engineering disciplines, are forced to fall back on mathematics to help us achieve this. Mathematics offers us the ability to unambiguously communicate requirements and offers us proof as a basis for a formal notion of correctness. But mathematics works only with formal models, so we might be able to prove a refinement relation exists between abstract and concrete versions of a model at a formal level, but the models at that level have to reflect the actual, informal world and its requirements and implementations. Thus we have a gap that we must narrow as far as we can:



To an engineer formality without interpretation is useless. Engineers must pay close attention to how to interpret formal models if they wish to stop planes falling out of the sky and other unwanted events from occurring in the actual but *informal* world around us.

Consider this 13^{th} century Sufi teaching story [1]:

> Once, a man found Mulla Nasruddin searching for something on the ground outside his house. On being asked, Nasruddin replied that he was looking for his key. The man also joined in the search and in due course asked Mulla: "Where exactly did you drop it?"
>
> Mulla answered: "In my house."
>
> "Then why are you looking here?" the man asked.
>
> "There is more light here than in my house," replied Mulla.

Theoreticians design languages and methodologies that, when successful, illuminate a path so engineers can construct reliable working software. But occasionally both theoreticians and engineers need to spend time looking, not in the bright light of the formal theory but in the shadowy world of its interpretation. The problem with interpretations is that by their very nature they are informal since they must always have informal components which "connect" with the actual world. Thus we can formally prove nothing about interpretations, no matter that it is important to make the correct interpretation and for interpretations to seem natural to the engineer.

Of course, what we do is to compile actual world collections of properties that we want our system to have and properties that we want it not to have, interpret these in our formal world and try to prove that the properties, so interpreted, that we want to hold do hold, and those that we do not want to hold do not hold. The more properties we can compile, and the more we can prove hold or do not hold, the more confidence we can have that our formalisation reflects the actual world. For any realistically large or complex system, this process can never be completed; the best we can do is make it as complete as possible.

In this paper we are interested in writing specifications and then constructing implementations that satisfy them. The formal construction of an implementation, or concrete specification, from a more abstract specification we will call a *refinement step*. Given that the very reason for writing the specification is to construct implementations that satisfy it we believe that it is very natural for the semantics of a specification to be intimately connected to the semantics of refinement. This is true for many event-based formal methodologies: in CSP failure semantics is intimately related to failures refinement. Failures refinement is not always satisfactory when non-terminating processes are considered. Consequently failures/divergences refinement [2], NDFD refinement [3] and CFFD refinement [3] have been defined, but in each case not only is a new refinement defined but also a new semantics is defined.

In some state-based formal methods the same semantics are defined (partial relations) but with several distinct refinement preorders [4, 5, 6]. Although there is nothing wrong with this state-based approach we will argue that it leaves some of the meaning of operations out of the formal semantics and may cause some difficulties.

Our approach will be to use a general parametrised framework, taken from [7, 8], where this intimate relation between semantics and refinement is a central idea and explore what effect it can have on a state-based formal method.

We are all familiar with mathematicians writing down terms that describe actual things and via (formal) reasoning drawing conclusions (other terms) from the original terms. It is the engineer who has the responsibility to interpret the terms and decide

whether the formal reasoning steps correctly reflect the informal world. We can help the engineers by defining a semantics for the terms that is closer to the engineers' interpretation of the world around them. Subsequently engineers only need think about the semantics rather than the terms. This works well as long as the correct semantics (and reasoning) is chosen.

Next we give a simple example to illustrate how easy it is to use the wrong semantics and how considering the formal model alone can not clarify the situation.

1.1 Example

As the truth of a statement is ascertained by the construction of a valid proof, reasoning about the proof of statements should be reliable. By reasoning about what can be proved we are going to offer a rigorous but informal argument that the following formal statement FS *is invalid*.

From assumption $(Pa \wedge Pb) \rightarrow R$ we can show $(Pa \rightarrow R) \vee (Pb \rightarrow R)$ **FS**

Informal Argument: The assumption is that from a proof of $Pa \wedge Pb$ we can construct a proof of R. But this does not necessarily mean that we can construct a proof of R just from a proof of Pa or construct a proof of R just from a proof of Pb. This is clearly true as knowing either Pa or Pb is to know less that to know both Pa and Pb and there is at least the possibility that the truth of both Pa and Pb were needed in the construction of the proof of R.

But despite this (hopefully) convincing informal argument we can provide a formal proof that from $(Pa \wedge Pb) \rightarrow R$ we can indeed show $(Pa \rightarrow R) \vee (Pb \rightarrow R)$.

1.	$(Pa \wedge Pb) \rightarrow R$	
2.	$\neg((Pa \rightarrow R) \vee (Pb \rightarrow R))$	Ass
3.	$\neg((\neg Pa \vee R) \vee (\neg Pb \vee R))$	$Def \rightarrow .2$
4.	$Pa \wedge \neg R \wedge Pb \wedge \neg R$	$DeMorgan, \neg E$
5.	$\neg(Pa \wedge Pb) \vee R$	$Def \rightarrow .1$
6.	$\neg(Pa \wedge Pb)$	Ass
7.	$\bot$	$From - 4, 6$
8.	R	Ass
9.	$\bot$	$From - 4, 8$
10.	$\bot$	$\vee E, 5, 6, 7, 8, 9$
11.	$(Pa \rightarrow R) \vee (Pb \rightarrow R)$	$Cont, 2, 10$

What has gone wrong?

1.2 Explanation

The first mistake in Section 1.1 is the assumption that the interpretation of formal statements is both obvious and universally agreed upon. Indeed the statement **FS** can be given a classical or a constructive [9, 10] interpretation.

With a constructive interpretation the informal argument is indeed correct. And the formal argument is incorrect as it is based upon classical logic. But if, as is common, the statement is given a classical logic interpretation then the mistake was made before the informal argument was constructed and indeed before the formal statement was given. The very first two sentences of Section 1.1 are mistaken. Classical logic is a

logic of *truth* (or at least truth as formalised by truth table semantics). It is constructive logic that is a logic of *proof* and hence choosing to base the argument on what can be *proved* is a mistake, as it makes use of the wrong semantics. Hence any informal argument based on proof cannot be said to relate to a classical interpretation of any formal statement. In particular the previously give informal argument does not relate to the classical interpretation of **FS**.

As using the semantics can so easily lead us astray it might be tempting to avoid it but this is not always very practical. To reason syntactically that one statement cannot be proved from another would require reasoning about all proofs which is not easy to do. In such situations it is usual to reason about a semantics and appeal to a soundness and completeness result.

This example illustrates that:

1. reasoning with semantics can be very helpful;
2. it is important to select the correct semantics; and
3. an apparently innocent change to the semantics, in the example the change is from *truth* to *proof*, can have disastrous effects.

One of the worst aspects of such "mistakes" is that they cannot be found by considering the formal arguments alone.

Naturally if changing the semantics is difficult then one solution is simply not to do it. But as illustrated later (Section 6) semantics are often changed even by theoreticians. It is useful to engineers and theoreticians alike to have different ways to safely interpret formal statements and choose the most appropriate interpretation for a given situation.

Here we are interested in the refinement of specifications and it is very clear from the many definitions in the literature there is certainly no one universally agreed upon definition of refinement or indeed one interpretation of what refinement means.

We will next look at how an engineer might informally interpret the statement that A is a refinement of C and provide some answers to the question: what use is a formal refinement to an engineer?

2 Interpretation and robustness of refinement

We are interested in refinement, that is in the formal transformation of an abstract specification into a more concrete specification.

Before we give our formalisation of refinement we look at three informal interpretations of refinement each based on an associated interpretation of a specification. Each of these different interpretations may be of use to the engineer in different situations.

Refinement interpreted as preservation of guarantee, $\mathsf{A} \sqsubseteq_p \mathsf{C}$ Under this interpretation a specification is interpreted as a *guarantee* that "if the entity is used in the prescribed way then one of the prescribed observable behaviours will be seen and nothing else". Then we have the following natural informal notion of refinement, which appears in many places in the literature [6, 4, 5, 11, 12, 13]

> The entity C is a refinement of a more abstract entity A when no user of A could observe if they were given C in place of A, which is to say that nothing they observe of C would suggest that they were not observing A, so the guarantee given with A is preserved.

Refinement interpreted as implication, $\mathsf{A} \sqsubseteq_{\rightarrow} \mathsf{C}$ Under this interpretation a specification is interpreted as an assertion about the behaviour of an entity (formalised naturally as a logical term $I_{\rightarrow}(\mathsf{A})$) [14]. A refinement relation holds between entities C and A if and only if the behaviours asserted by the interpretation of C satisfy the interpretation of A, formalised by $\mathsf{A} \sqsubseteq_{\rightarrow} \mathsf{C}$ iff $I_{\rightarrow}(\mathsf{C}) \rightarrow I_{\rightarrow}(\mathsf{A})$.

Refinement interpreted as subset of implementations, $\mathsf{A} \sqsubseteq_{i} \mathsf{C}$ Under this interpretation a specification is given by the set of its implementations. A concrete specification is a refinement of a more abstract specification if and only if the implementations satisfying the concrete specification are a subset of the implementations satisfying the abstract specification.

Definition 1 *Two interpretations of refinement* $\sqsubseteq_x$ *and* $\sqsubseteq_y$ *are called consistent if and only if for all entities* A *and* C $(\mathsf{A} \sqsubseteq_x \mathsf{C}) \leftrightarrow (\mathsf{A} \sqsubseteq_y \mathsf{C})$.

Definition 2 *The more interpretations a formal definition of refinement has that are consistent with each other, the more robust is the definition.*

In what follows, we advocate robustness both because it is useful to have different ways to interpret (and formalise) the same intuition (about what refinement is) and because we believe errors with the semantics (of refinement) are less likely to occur given a robust definition characterising the intuition.

We also see that, in the three sorts of refinement listed above, the refinement preorder characterises the semantics of a specification, and vice versa. This is another way in which a definition of refinement and a semantics of specifications can both be regarded as very robust.

The idea that refinement and specification should characterise each other is not a new idea, nor is the usefulness of having more than one semantics. In the event-based world it would seem strange to use failure semantics and not use failure refinement, although it would be possible to do this. Matthew Hennessy [15] constructs an elegant trio of semantic and refinement definitions, axiomatic, testing and denotational, and formally proves that are all consistent. He then goes on to consider different refinement preorders, but for each he constructs a different axiomatic, testing and denotational semantics.

In the event-based literature the definition of refinement frequently characterises the denotational semantics and hence it is not uncommon to use the definition of refinement to define the meaning or denotation of the operational semantics. This has been so popular an approach that a survey of over 150 different semantics, all based on different definitions of refinement, can be found in [16, 17].

In the state-based literature this approach is not so common, but what is common is to define the semantics of an operation as a partial relation. Then different definitions of refinement, based on different interpretations of the partial relations, can be given

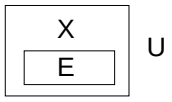


Figure 1: Entity, conteXt and User and their interfaces

but without changing the semantics. In Section 6 we will discuss some consequences of this approach that could be avoided by using a semantic definition that, as in event-based approaches, is closely related to a robust definition of refinement.

3 A robust interpretation of refinement

This section gives an outline of a formal definition of refinement and three consistent interpretations that follow from it (for further details see [7, 8]).

Our first step towards formalising refinement is to decide what the user can observe, so we make some assumptions. In practice we are interested in reasoning about and refining small entities (modules) which are combined to make a larger entity. Thus we model an entity E as existing in some context X (the rest of the larger whole) interacting on the set of actions Act. All E's actions interact with X at the E-X interface (see Figure 1). X and U interact at a different interface and on a disjoint set of actions. We model the observer as a *passive* user U that is a third entity that observes or interacts with X, but neither blocks the X actions nor interferes with E-X communication.

We will give formal general definitions of refinement with explicit parameters representing both Ξ, the contexts in which entities will be placed, and O, an observation function from entities to sets of traces from $\wp(\mathbb{O})$ (of event names or states), where each trace $tr \in \mathbb{O}$ is a potential observation.

This general model can be made more concrete by instantiating its parameters defining : one, how we represent our entities; two, the sets of contexts Ξ; and three, the observation function O from entities to sets of traces.

This results in what we call a *special theory*. It has been shown ([18]) that some of the classic theories of operations, abstract data types (ADT) and processes that appear in the literature are special theories of the general model given here.

3.1 Refinement interpreted as preservation of guarantee

Definition 3 *Let Ξ be a set of contexts each of which the entities* A *and* C *can communicate privately with, and let O be a function which returns a set of traces, each trace being what a user observes of an execution. Then*[1]

$$\mathsf{A} \sqsubseteq_{\Xi,O} \mathsf{C} \triangleq \forall x \in \Xi . O([\mathsf{C}]_x) \subseteq O([\mathsf{A}]_x)$$

[1] $[\mathsf{E}]_x$ denotes the execution of entity E in context x.

3.2 Refinement as implication

It is easy to see that we can give entities in our general model a relational semantics. We are not the first to use relations as a semantics for a diverse range of models: indeed Hoare and He in their Unifying Theories of Programming (UTP, [14]) do just this. The main difference between this work and others is that we motivate our relational semantics by defining a consistent testing semantics.

Definition 4 *Let Ξ be a set of contexts each of which the entity A can communicate privately with, and O be a function which returns a set of traces, each trace being what a user might observe of an execution. The relational semantics of an entity A is a subset of $\Xi \times \mathbb{O}$. Let*

$$\mathsf{A}_{\Xi,O}(x,o) \triangleq x \in \Xi \wedge o \in O([\mathsf{A}]_x)$$

then

$$[\![\mathsf{A}]\!]_{\Xi,O} \triangleq \{(x,o) | \mathsf{A}_{\Xi,O}(x,o)\}$$

Refinement is now the subset relation between relations or implication between the predicates that define them.

For any entities A and C let

$$\mathsf{A} \sqsubseteq_{\rightarrow,\Xi,O} \mathsf{C} \triangleq \mathsf{C}_{\Xi,O} \rightarrow \mathsf{A}_{\Xi,O}$$

then we have our first consistency result:

$$\mathsf{A} \sqsubseteq_{\Xi,O} \mathsf{C} \Leftrightarrow [\![\mathsf{C}]\!]_{\Xi,O} \subseteq [\![\mathsf{A}]\!]_{\Xi,O} \Leftrightarrow \mathsf{A} \sqsubseteq_{\rightarrow,\Xi,O} \mathsf{C}$$

3.3 Refinement as subset of implementation

Given that refinement is frequently characterised as the reduction of non-determinism and that software is currently run on deterministic computers we will define what it means to be deterministic in our general model.

Definition 5 *An entity A is deterministic iff its relational semantics is a function:*

$$Det_{\Xi,O}(\mathsf{A}) \triangleq (x,o) \in [\![\mathsf{A}]\!]_{\Xi,O} \wedge (x,p) \in [\![\mathsf{A}]\!]_{\Xi,O} \Rightarrow o = p$$

We now say that *implementations are deterministic entities* and so we can define the semantics of an entity A to be the set of implementations that satisfy it:

$$[\![\mathsf{A}]\!]_{I,\Xi,O} \triangleq \{(\mathsf{I} | Det_{\Xi,O}(\mathsf{I}) \wedge [\![\mathsf{I}]\!]_{\Xi,O} \subseteq [\![\mathsf{A}]\!]_{\Xi,O}\}$$

and then for any entities A and C:

$$\mathsf{A} \sqsubseteq_{I,\Xi,O} \mathsf{C} \triangleq \mathsf{C}_{I,\Xi,O} \subseteq \mathsf{A}_{I,\Xi,O}$$

Using this we recreate the relational semantics of the entity by taking the union of the functions and see our second consistency result:

$$\mathsf{A} \sqsubseteq_{I,\Xi,O} \mathsf{C} \Leftrightarrow \mathsf{A} \sqsubseteq_{\Xi,O} \mathsf{C}$$

4 Interfaces

In this section we will show why both contexts and users are needed to define refinement by demonstrating situations where two different types of interfaces are needed: a *transactional* interface between entities and contexts and an *interactive* interface between contexts and users.

We will refer to an interface as *transactional* if interaction (observation) occurs at no more than two distinct points: initialisation and finalisation of the entity. If termination is successful then there may be distinct observations that could be made at finalisation, but if termination is unsuccessful then all that can be "observed" is that the entity fails to terminate.

An example of an entity with transactional interaction is a program that accepts a parameter when called and returns a value when it terminates. Clearly if the program fails to terminate no value can be returned.

In contrast we refer to an interface as *interactive* when interaction can occur at many points throughout the execution. Hence with interactive interfaces more than one observation can be made prior to termination and even prior to non-termination.

An example of an interactive entity is a coffee machine. To obtain two cups of coffee the user first inserts a coin, then pushes the appropriate button and takes the first cup of coffee. But if after inserting a second coin the vending machine now "fails to terminate" by not producing a second cup of coffee the previously successful interactions mean that what has been observed cannot be represented by noting non-termination alone. (We still have our first cup of coffee!)

5 Abstract Data Type Refinement

With entities being abstract data types, contexts being the programs that use the ADTs (by using, calling, the operations the ADT provides) and users being the users of the program, we must have an interactive ADT/program interface. But the definition of refinement is sensitive to the type of program - user interface (see [7] for details).

A computational method for deciding whether data refinement holds between ADTs is problematic as the definition of data refinement involves quantification over all programs (usually an infinite collection). But the classic Hoare, He and Saunders result [19] uses retrieve or simulation relations between the state spaces of two the ADTs to define a forward or backward simulation between them. Usefully, the simulations are quantified only over all operations (a finite collection) in the ADTs, and it is proved that they are sound and jointly complete with respect to refinement. Thus the Hoare, He and Saunders guarantee is that if A is a forward or backward simulation of C then any observation that can be made of any program using C could have been made of the same program using A.

The Hoare, He and Saunders proof is based on the operations having a relational semantics and the behaviour of the program under consideration being defined by sequential composition of the relational semantics of individual operations. The proof makes no restriction on the relations used to model the operations and to define the retrieve or simulation relation.

Partial relations are open to a variety of interpretations (see [5, 6, 4] for three distinct interpretations). For example, let $D \triangleq \{a, b\}$ be a state space of two states and take an operation P where $[\![\mathsf{P}]\!] \subseteq D \times D$ define the semantics of P by $[\![\mathsf{P}]\!] \triangleq \{(a, a)\}$. This specification can be considered either as requiring a partially correct implementation: when an implementation of P is started from state a then if it terminates it will terminate in state a; or as requiring a totally correct implementation: when an implementation of operation P is started from state a then it will terminate and it will terminate in state a. In addition the implementation's behaviour from state b could be interpreted as undefined or as blocked.

We need to be wary of using the sequential composition of partial relations since as Spivey pointed out modelling sequential composition of operations as the relational composition of partial relations has a meaning that *"differs from the meaning that would be be natural in a programming language"*, Spivey [20, p136].

For example let $[\![\mathsf{O}]\!] \triangleq \{(a, a)(a, b)\}$ and $[\![\mathsf{P}]\!] \triangleq \{(a, a)\}$ and let $[\![\mathsf{O}; \mathsf{P}]\!] \triangleq [\![\mathsf{O}]\!]; [\![\mathsf{P}]\!]$. Spivey's problem can be seen by considering $[\![\mathsf{O}; \mathsf{P}]\!] = \{(a, a)\}$ and asking what has happened when an implementation of O terminates in state b.

It is easy to see that modelling the relational semantics of a sequence of operations as the sequential composition of the relational semantics of the individual operations is consistent with a partial correctness interpretation but not consistent with a total correctness interpretation . The Hoare, He and Saunders results is based on the relational semantics of a sequence of operations being the sequential composition of the relational semantics of the operations. Thus to avoid Spivey's problem operations with a partial relation semantics must have a partial correctness interpretation[2].

6 Semantic changes, the benign and the problematic

In the state based world Z, B and Event B use partial relations as their operational semantics. Thus Z, B and Event B formal models are of interest to us as they possess a formal operational semantics and yet are, by design, open to a variety of interpretations.

This has the desirable consequence of allowing these methods to be flexible in that they can be used in a wide range of situations. Also, though this opens up the possibility that problems, of the kind discussed in Section 1 might be introduced.

Experts in Z know only too well that, given some common interpretations of Z, to use Z safely (avoiding the problems discussed in Section 5) you need to restrict how it is used to a particular (informal) methodology, a good example of which is the informal method followed in [5].

B [13] is like Z but with its methodology formally built into the B tool kit. The relational semantics of a B operation is, in part of its domain, totalised so as to be undefined. But this happens only in part of its domain; the relational semantics is still a partial relation. Then, prior to computing refinement of a B machine it is required that all operations be proved total (see [13, p297 Property 6.4.1, p525 Condition1]) on some domain D and that it be proved that only states in D are reachable. Thus B has

[2]To readers familiar with [4] this may seem so obvious that it is not worth stating but to casual readers familiar with [5] this may even seem untrue. Consequently this point will be discussed in more detail in Section 6.1.

a two stage development: initially operations with partial relational semantics can be defined but when a machine is to be refined Spivey's problem can be ignored as only operations with a total relational semantics need be considered.

Event B is more like Z in that it is, by design, open to many interpretations and Spivey's problem is not avoided by any formal methodology. Most notably Event B refinement is defined to be (only) the existence of forward simulation and not the "full" refinement as defined in B [13] and elsewhere [5, 19, 4, 6]. Interesting questions that arise from this are: what kind of contexts are Event B machines designed to operate in?; and what guarantee does Event B refinement offer the engineer?

6.1 Data types or Processes

Woodcock and Davies' definition of data refinement in [5] is that taken from [19] and applied to ADT with operations that are interpreted as undefined outside of precondition, which some call contractual. Although using Z, with its partial relation semantics, they define data refinement while using a total correctness interpretation of the relational semantics. They have achieved this by changing the semantics of operations from Z partial relations to a lifted totalled semantics prior to computing data refinement. Another view is that the semantics has not changed and the transformation to total relations is just a step in the computation of refinement. Whatever your view, the transformation exists. For ease of discussion we will refer to it as a transformation of the semantics.

The engineer must therefore bear in mind that the Hoare, He and Saunders guarantee applied to the data types in [5] is based on operations with a total relation semantics due to the transformation of the semantics, not on Z's "official" partial relation semantics.

In [5, Table 16.1] value passing operations are modelled my winding the values input into a sequence initialised at the start of the program execution and the output values wound into a sequence to be observed. In addition another set of rules [5, Table 16.2] is defined in which input and output occurs and can be seen at each step.

In our terminology there is, therefore, a change in the program/user interface from transactional to interactive, and thus a change in the definition of refinement. Because we advocate using robust definitions of refinement where refinement can be used to define the semantics, we choose to view this change of refinement as a second transformation in the semantics. However, this is benign, as the two sets of rules are apparently equivalent.

Subsequently, Bolton and Davies' definition of data refinement in [6] is also that taken from [19] but is applied to ADTs with operations that are interpreted as blocked or guarded outside of precondition, which some call behavioural. They go on to make similar semantic transformations to those in [5]. But this time the second semantic transformation, that of changing the program/user interface from transactional to interactive, results in a subtly different refinement relation for which backward simulation *is not sound* [21]. One of the difficulties is that having made the apparently benign transformation in the semantics the lack of soundness cannot be discovered simply by looking at the formality. Just as in our example Section 1.1, looking at the formal proof alone will not reveal any errors.

6.2 Relational semantics or Logical semantics

The initial semantic transformation in Section 6.1 replaces partial relations with lifted total relations. The logical or axiomatic approach provides an alternative to using a transformation because it keeps the partial relation semantics but defines sets of axioms to characterise refinement.

Which semantics, the relational or logical, was used was regarded as unimportant as any refinement based on the logical definition was also, it was assumed, a refinement based on the relational definition.

But recently Boiten and Derrick have shown [22] a the key result: the completeness of forward and backward simulation with respect to data refinement fails to hold for operations that are blocked outside of precondition when their semantics is given in the logical style, although it still holds for the relational-style semantics [21]. Further, using a *restricted* simulation relation, a soundness and completeness result can be re-established [21] for the logical style. This again shows us that an apparently innocent change in semantics can have unforeseen consequences.

If we regard Z's semantics as given by partial relations (where outside of precondition a specification is necessarily silent about what happens due to the partiality), yet use an axiomatic or logical definition of refinement, based for example on "undefined outside of precondition", then we have a semantics that is silent about what happens outside of precondition but a definition of refinement that is not silent about what happens outside of precondition. We can view this as saying that the definition of refinement extends the semantics with additional meaning, not found in the semantics (in this case the additional meaning defines the behaviour outside of the precondition).

It is now not clear if the meaning of the specification is given by the semantics or by the definition of refinement. If the meaning is given by the semantics then why use this definition of refinement that is based on a different meaning? If the meaning is given by refinement then why not formalise this in the semantics?

A way to make the formal model more robust would be, as we have advocated, to change the semantics to keep it "consistent" with the definition of refinement.

6.3 Conclusions

From the previous two sections and our example in Section 1.1 we conclude that understanding or interpreting formal models to the extent required to prevent failure of real (software) systems is far from easy. Apparently innocent changes to the semantic model can very easily introduce errors that are hard to detect even by theoreticians. Despite the difficulty of designing safe, useful theoretical frameworks we still need to give engineers greater freedom in how they develop software.

7 Stepwise design

To design reliable complex systems that are open to human understanding we need both simple and intuitive high-level descriptions that clearly reflect the required behaviour and detailed low-level descriptions that an implementation can be clearly seen

to satisfy.

If we tried to specify everything down to the last detail early in the design process we would fail to see any clear, big picture. Consequently we wish to add detail in a stepwise fashion through out the design process. But we wish to avoid leaving informal any essential methodological restrictions, so we wish to follow the design of B and formalise, as much as we can, any essential methodology.

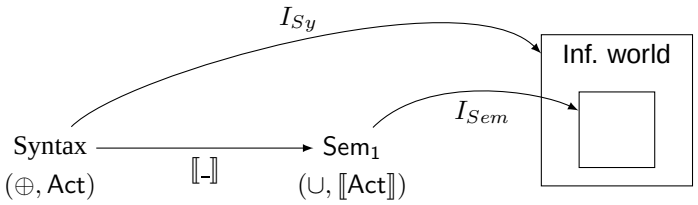
7.1 Stepwise semantics

The advantage of basing a wide variety of semantic models all on one common semantics is that we can uniformly define some operations, such as parallel composition, choice, event hiding, recursion on the semantics and then, with some effort, lift these definitions to the more detailed semantic models. For example, [16, 17] and referred to earlier, has one definition of parallel composition not 150 definitions of parallel composition.

We advocate the following steps towards a target semantics.

Step 1. Any formal statement is written using some well-defined syntax. For example, let there be terms T_Σ constructed from some signature $\Sigma \triangleq \{\oplus\} \cup \mathsf{Act}$, where Act is a set of actions of interest, and $\mathsf{Act} \subseteq \mathsf{A}$, where A is the set of all possible actions, and $\oplus$ is an in-fixed binary operator $\oplus \in \mathsf{A} \times \mathsf{A} \rightarrow \mathsf{A}$.

To be of use to an engineer this must have some interpretation in the informal world where they work and do their modelling and designing. Such an interpretation I_{Sy} of terms in our example language is very flexible in as much as the terms can be interpreted as representing any entity from a set of things with a binary operation on this set. This syntax puts no further restrictions on what interpretations can be made.



We can reduce the flexibility in the way the terms can be interpreted by specifying a formal semantics for them. So, continuing our example, let us define the formal semantics of an action a to be a relation $[\![\mathsf{a}]\!] \subseteq S \times S$ over some set S, $[\![\mathsf{Act}]\!] \triangleq \{[\![\mathsf{a}]\!] | \mathsf{a} \in \mathsf{Act}\}$ and the semantics of the binary operator to be set union $[\![\oplus]\!] \triangleq \cup$.

From the semantic interpretation we can infer and equation: $[\![\mathsf{a} \oplus \mathsf{a}]\!] = [\![\mathsf{a}]\!]$. So now the valid interpretations are restricted to a subset of the valid interpretations given by I_{Sy}, namely by eliminating those that do not obey the equation.

We will write I_{Sem} for the standard and obvious informal interpretation of S as some set of states, a in Act as being an operation which moves between states and $[\![\mathsf{a}]\!]$ as the state-to-state relational semantics of the operation a. I_{Sem} is a valid interpretation for this more restricted semantics since is obeys the equation. And of course I_{Sem} talks about less of the informal world than I_{Sy} did, as our diagram suggests.

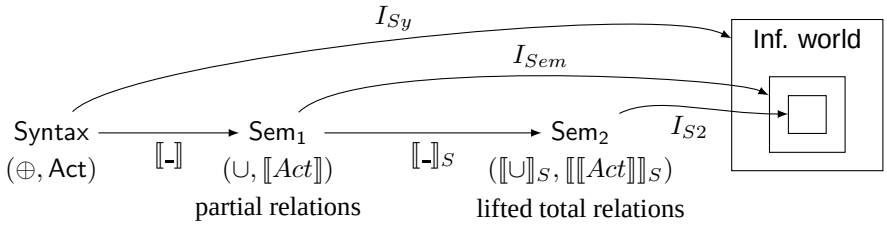
Let I_A and $I_C.[\![\_]\!]$ be informal mappings from some formal domain D to the real world. We will refer to I_C as a I-refinement of I_A when for all d in D, $I_C([\![d]\!])$ is a subset of $I_A(d)$.

We have used some English here rather than using only mathematical notation to remind the reader that this has to be an informal definition (the informal world is involved), but from now we will rely on the reader to remember that all interpretations are an informal mapping into the informal ("real") world. Clearly in our example $I_{Sem}([\![d]\!])$ is a subset of $I_{Sy}(d)$ and hence I_{Sem} is an I-refinement of I_{Sy}.

To help make intuitions more robust we require that:

1. interpretations are homomorphic, e.g. $I_{Sy}(\mathsf{a} \oplus \mathsf{b}) \triangleq I_{Sy}(\mathsf{a})I_{Sy}(\oplus)I_{Sy}(\mathsf{b})$ and $I_{Sem}(\mathsf{a} \oplus \mathsf{b}) \triangleq I_{Sem}(\mathsf{a})I_{Sem}(\oplus)I_{Sem}(\mathsf{b})$
2. so are semantic mappings, $[\![\_]\!]$ the semantics of a term is given by the semantics of its components, e.g. $[\![\mathsf{a} \oplus \mathsf{b}]\!] \triangleq [\![\mathsf{a}]\!][\![\oplus]\!][\![\mathsf{b}]\!]$
3. informal intuitions are preserved, $I_{Sy}(\oplus) = I_{Sem}([\![\oplus]\!])$

Step 2. Let us extend our example and assume that $[\![\mathsf{a}]\!] = \{(1,1)\}$ and the state space is given by $S = \{1,2\}$. This relational semantics can be interpreted in several different ways, see Section 5 for four distinct interpretations.



Just as we refined how we interpret the syntax by defining the semantics of the terms we can also refine how we interpret the initial semantics Sem_1 by defining a meaning (second semantics, Sem_2) for the initial semantics. In our example we can define how to lift and totalise the initial partial relation semantics. Lifting adds $\perp$ to S to give $S_\perp$ and operations now have $S_\perp \times S_\perp$ relational semantics and $\perp$ on the left of the relation is interpreted as the operation fails to start and on the right of the relation it is interpreted as the operation fails to terminate[3]. How we totalise the relation formalises the interpretation we wish to give it.

Interpreting $[\![\mathsf{a}]\!] = \{(1,1)\}$ as blocked (guarded) outside of precondition and requiring a totally correct implementation (so it must terminate from state 1) we map all, and only, states outside of the precondition to $\perp$ and only to $\perp$. Thus we have $[\![\{(1,1)\}]\!]_S = \{(1,1),(2,\perp),(\perp,\perp)\}$.

In our example the meaning of the partial relation semantics is has been formalised by the application of a semantic function $[\![\_]\!]_S$ that lifts and makes total the partial

[3]For details of how to interpret the usual pre state, post state relations as relations between contexts and observation traces see [7, 21] and for details of how to extend this interpretation to cover lifted relations $S_\perp \times S_\perp$ see [21]

relations. Of course we do not need to go through the intermediate semantics (partial relations) we could simply use a mapping $[\![[\![\_]\!]]\!]_S$ from the syntax to the new semantics. The advantage of using an intermediate semantics is that mathematical definitions and results can be established for the the initial, or intermediate, semantics and this used to establish similar results for a whole range of more detailed semantics.

Because care is needed to make sure that intuitions at a high level of abstraction, for example with partial relation semantics $[\![\_]\!]$, transfer correctly to a less abstract level, for example for the lifted totalised semantics $[\![\_]\!]_S$ we advocate keeping to the three points raised at the end of step 1.

7.2 Refining interpretations

Applying stepwise design to one of our robust interpretations of a high-level refinement, as defined in Section 3, can be done by including an explicit refinement operator $\sqsubseteq_H$ in the signature of our terms in the definition of our syntax in step 1. This allows us to talk about refinement at some level of abstraction, or equivalently gives a theory of refinement at some level of abstraction. We can now interpret this theory as a distinct further theory based at another level of abstraction. We will often use this method to view the original refinement in the original theory as taking place at a high-level of abstraction and the further theory given by the interpretation of the high-level theory as giving us a lower-level theory with its own lower-level refinement , which we will call $\sqsubseteq_L$ (see [8] for more details). This interpretation between theories is formalised by defining two semantic mappings. We use a semantic mapping $[\![\_]\!]_v$ to interpret, or *embed*, high-level $\mathsf{E_H}$ entities as low-level entities $\mathsf{E_L}$ and a separate semantic mapping vA to interpret, or *embed*, low-level entities as high-level entities. When they form a Galois connection we call such pairs of semantic mappings a *vertical refinement*, denoted by $\sqsubseteq_v$, and write $\mathsf{E_H} \sqsubseteq_v [\![\mathsf{E_H}]\!]_v$

In Section 3.2 we have refinement as implication and we can view the context and observation function pair from that section as defining a logical theory and then apply the well-known reading of Galois connections as theory transformations between two theories, one at a high level based on $(\Xi_\mathsf{H}, O_\mathsf{H})$ and the other at a lower level based on $(\Xi_\mathsf{L}, O_\mathsf{L})$. Galois connections thus provide a very strict design step between theories and preserve many features of the theories including union, subset (which we use to define refinement), and fixed points.

For our purposes all we need consider are simple Galois connections such as the subset morphisms $[\![\_]\!]_\supseteq$ where $\mathsf{E_H} \subseteq [\![\mathsf{E_H}]\!]_\subseteq$ and hence $\mathsf{E_H} \sqsubseteq_\supseteq \mathsf{E_L}$ implies $\mathsf{E_H} \subseteq \mathsf{E_L}$. Intuitively we can think of $(\Xi_\mathsf{H}, O_\mathsf{H})$ as defining a *frame* outside of which the high-level (abstract) specification is *silent*. We note that we can use *silent outside of frame* to give yet another valid interpretation to Z's partial relational semantics.

The simple version of vertical refinement with subset morphisms is able to introduce nondeterminism, outside of frame, unlike one of our refinements defined in Section 3 which never introduce nondeterminism. Nevertheless, we call it a refinement because it offers an engineer a simple guarantee: that any behaviour of the low-level (concrete) specification that lies within the frame is a behaviour of the high-level (abstract) specification is behaviour of the abstract specification.

What happens to Spivey's problem and the lack of monotonicity when we use robust definitions of refinement and semantics? We are not attempting to offer a magic solution to these problems because we believe there are none. Assume we start with a robust definition of semantics and refinement where the semantics are partial relations. Recall that fixing the refinement fixes the semantics, so if we try to change the refinement to formalise the behaviour outside the precondition, e.g to being undefined, or to being guarded, then we are forced to change the semantics. We can change the semantics by constructing a Galois connection between two theories. This is where our approach stops us relying on informal methodology as we now explain.

If Z_{op} is a Z operation schema it has a partial relation semantics $\mathsf{R_{op}}$ with domain $dom(\mathsf{R_{op}})$ and range $ran(\mathsf{R_{op}})$. From this we can define the contexts $\Xi_{\mathsf{op}} = dom(\mathsf{R_{op}})$ and observations $\{(a,b)|a \in dom(\mathsf{R_{op}}) \wedge b \in ran(\mathsf{R_{op}})\}$. Thus different operation schemas exist on different layers, or in different theories, and we make use of Galois connections to relate one theory with another. We have not yet considered relating a set of theories to a single theory, and indeed it is unclear how we would interpret a set of theories as a single thing. Hence we have not yet attempted to relate partial relations as a whole to any theory

A well-known solution to these problems, that B [13] adopts, is to restrict refinement and sequential composition to being applied only when all operations are total on some domain D and the operations never leave D. Adopting this solution we can restrict the operational semantics to total relations over $D \times D$. Thus all operations now exist in the same theory or layer. With this restriction, proved to be useful in practice, it is easy to establish that: Spivey's problem no longer applies, monotonicity results hold and $[\![\mathsf{a} \cup \mathsf{b}]\!]_S \triangleq [\![\mathsf{a}]\!]_S \cup [\![\mathsf{b}]\!]_S$ is true when $[\![\cup]\!]_S \triangleq \cup$.

What we have ended up with is a very familiar two-step approach: first, reason about partial relations and avoid refinement and sequencing; and, secondly, only when these partial relations have been used to build total relations do we apply refinement and sequencing. We make no claim for novelty here as it can agued that it appears in the B tool kit, in the informal methodology of Using Z [5] and even in Dijkstra's early work [23]

8 Conclusion

The use of robust definitions of semantics and refinement as favoured in the event-based literature has been used as the basis for a state-based approach that keeps track of what is in the formal model and what remains to be interpreted informally. We advocate three broad principles:

One define refinement and the semantics of specifications to be robust

Two even small changes to a formal semantics should be checked formally

Three in stepwise design the semantic mappings should respect how specifications are composed by their operators

1. the semantics of a term is built from the semantics of its components

2. our informal intuitions are preserved

By following these principles we have interpreted partial relations as silent outside of frame and only if we consider operations that are all total on some domain have we been able to proceed by formal stepwise development.

References

[1] Ltd, T.P.: The Fabulous Adventures of Nasruddin Hoja. (Ta-Ha Publishers Ltd. (UK))

[2] Roscoe, A.: The Theory and Practice of Concurrency. Prentice Hall International Series in Computer Science (1997)

[3] Valmari, A., Tienari, M.: Compositional Failure-based Semantics Models for Basic LOTOS. Formal Aspects of Computing **7** (1995) 440–468

[4] de Roever, W.P., Engelhardt, K.: Data Refinement: Model-Oriented Proof Methods and their Comparison. Cambridge Tracts in Theoretical Computer Science 47. Cambridge University Press (1998)

[5] Woodcock, J., Davies, J.: Using Z: Specification, Refinement and Proof. Prentice Hall (1996)

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

[7] Reeves, S., Streader, D.: General refinement, part one: interfaces, determinism and special refinement. In: Refine08 - International Refinement Workshop, Turku, Elsevier (2008) to appear.

[8] Reeves, S., Streader, D.: General refinement, part two: flexible. In: Refine08 - International Refinement Workshop, Turku, Elsevier (2008) to appear.

[9] Troelstra, A.S.: From constructivism to computer science. Theor. Comput. Sci. **211** (1999) 233–252

[10] Bridges, D., Reeves, S.: Constructive Mathematics in Theory and Programming Practice. Philosophia Mathematica **7** (1999) 65–104

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

[12] Derrick, J., Boiten, E.: Refinement in Z and Object-Z: Foundations and Advanced Applications. Formal Approaches to Computing and Information Technology. Springer (2001)

[13] Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)

[14] Hoare, C., Jifeng, H.: Unifying Theories of Programming. Prentice Hall International Series in Computer Science (1998)

[15] Hennessy, M.: Algebraic Theory of Processes. The MIT Press (1988)

[16] van Glabbeek, R.J.: Linear Time-Branching Time Spectrum I. In: CONCUR '90 Theories of Concurrency: Unification and Extension. LNCS 458, Springer-Verlag (1990) 278–297

[17] van Glabbeek, R.J.: The Linear Time - Branching Time Spectrum II. In: International Conference on Concurrency Theory. (1993) 66–81

[18] Reeves, S., Streader, D.: State- and Event-based refinement. Technical report, University of Waikato (2006) Computer Science Working Paper Series 09/2006, ISSN 1170-487X, http://researchcommons.waikato.ac.nz/cms_papers/12/.

[19] He, J., Hoare, C., Sanders, J.: Data refinement refined. ESOP 86 LNCS **213** (1986) 187–196

[20] Spivey, J.M.: The Z notation: A reference manual. 2nd. edn. Prentice Hall (1992)

[21] Reeves, S., Streader, D.: Guarded operations Refinement and Simulation. Technical report, University of Waikato (2009) Computer Science Technical Report 0–/2009 , http://www.cs.waikato.ac.nz/∼dstr.

[22] Boiten, E., Derrick, J.: Incompleteness of relational simulations in the blocking paradigm. In ?? **??** (2008)

[23] Dijkstra, E.W.: A Discipline of Programming. Prentice Hall (1976)