Recasting Hoare and He's Unifying Theory of Programs in the Context of General Correctness

Steve Dunne

School of Computing and Mathematics, University of Teesside
Middlesbrough, TS1 3BA, UK
email: s.e.dunne@tees.ac.uk

Abstract. We review Hoare and He's single-predicate model of sequential programs, and uncover its inability always to provide an adequate description of the required behaviour of a sequential program which implements a partial decision procedure: a significant shortcoming in an interactive era where such programs are often found as components of reactive systems. We show how the single-predicate model can be recast to overcome this limitation, using a variation on the predicate syntactic form employed by Hoare and He for their designs, which we call a *prescription*. We show that our prescriptions have a remarkably simple semantic characterisation by means of a single intuitively compelling healthiness condition. Our model also admits a pleasingly simple algebraic formulation which reveals it as a natural completion of Hoare and He's model; indeed, we can show the latter to be congruent to a restricted sub-model of the former.

1 Introduction

In [7] Hoare and He are concerned to identify for their relational theory of programs a subclass of relations which represent implementable programs. At the beginning of [7] chapter 3 they reveal their particular interest in a subclass of relations P which can be proved to satisfy these zero laws of sequential composition:

$$\text{true} \,;\, P \quad = \quad \text{true} \quad = \quad P \,;\, \text{true}$$

We quote from [7]:

> The easiest way to define the required subclass is to use the [zero] laws themselves as the defining property. Unfortunately the class of relations that satisfy the zero laws themselves is too large, and does not have the right closure properties. It is necessary to define a slightly more restricted set of predicates by means of a slightly stronger collection of laws known as healthiness conditions ... H1, H2, H3 and H4.

One almost discerns that the authors are disappointed not to be able to offer a purely algebraic characterisation of their subclass of relations. Indeed, when

we examine the healthiness conditions H1, H2, H3 and H4 in section 4.3 we will see that Hoare and He came very close to achieving their ideal of characterising implementable relations entirely in terms of, if not zero laws alone, then at least zero and unit laws: H3 is already a right unit law and H4 is a right zero law; although H1 doesn't seem to have such a form, [7] theorem 3.2.2 in fact shows that a predicate is H1 if and only if it satisfies a left zero and a left unit law. They were thwarted only by their H2 which cannot be rendered in such a way.

In fact, Hoare and He's initial instincts in favour of a completely algebraic characterisation turn out to be more well-founded than they concluded. But they focused their attention exclusively on programs whose behaviour is specified purely in total correctness, and so overlooked a different class of relations, which more faithfully model the actual behaviour of programs in the finer-grained semantics of general correctness. Such a new class of relations admits precisely the kind of algebraic characterisation whose infeasibility Hoare and He reluctantly had to concede for their own. The contribution of this paper is to illuminate this new class of general-correctness relations and present its complete semantic characterisation.

The rest of the paper is organised as follows: section 2 presents some basic definitions and lemmas concerning predicates which we utilise throughout the paper; sections 3 and 4 contain nothing new; they merely summarise Hoare and He's existing relational model, so as to make this paper conveniently self-contained even for a reader not previously familiar with that work, and to provide a relevant context for our work in the succeeding sections; in section 5 we identify a particular semantic inadequacy in the Hoare-He model which motivates our work; in the succeeding sections after section 5 we progressively develop our new relational model of sequential programs, along the way establishing the various theorems which underpin it.

2 Some Preliminaries concerning Predicates

We will deal extensively with predicates over an alphabet of variables which includes boolean-valued variables like ok and ok'. It is therefore useful to describe here some of the basic concepts and properties of such predicates, upon which later we will frequently rely.

We define an implication ordering $\sqsubseteq$ for predicates over a given alphabet: let A and B be predicates over an alphabet comprising a list of variables α; then we have

Definition 1 (Implication ordering)

$$A \sqsubseteq B \quad =_{df} \quad \forall \alpha \bullet B \Rightarrow A$$

The constant predicates true and false are respectively bottom and top of this ordering. For any predicate P it follows that $\mathrm{true} \sqsubseteq P \sqsubseteq \mathrm{false}$.

We can now define our fundamental notion of equality between predicates: let A and B be predicates over the same alphabet; then we have

Definition 2 (Equality for predicates)

$$A \;=\; B \quad =_{df} \quad (A \sqsubseteq B) \;\wedge\; (B \sqsubseteq A)$$

We use a substitution notation in connection with our predicates: if P is a predicate then $P[E/x]$ denotes the predicate derived from P by syntactically replacing each free occurrence of the variable x by the expression E.

Lemma 1 (Equality by cases)

If A and B are predicates over an alphabet of variables which includes a boolean-valued variable b, then $A = B$ if and only if

$$A[\text{true}/b] \;=\; B[\text{true}/b] \qquad \text{and} \qquad A[\text{false}/b] \;=\; B[\mathit{false}/b]$$

Proof: by definition of $=$ for predicates, which involves universal quantification over all variables of the alphabet, including b whose range of values is the boolean domain {true, false}. So we can separate out in particular the universal quantification over b and express the original equality as the conjunction of the two new equalities derived from these two cases for b.

□

Lemma 2 (Properties of predicates)

If B is a predicate over an alphabet of variables which includes a boolean-valued variable b, then the following properties hold:

$$(2.1) \qquad B \;=\; (b \Rightarrow B[\text{true}/b]) \;\wedge\; (\neg\, b \Rightarrow B[\text{false}/b])$$

$$(2.2) \qquad B \;=\; (b \wedge B[\text{true}/b]) \;\vee\; (\neg\, b \wedge B[\text{false}/b])$$

$$(2.3) \qquad b \wedge B \;=\; b \wedge B[\mathit{true}/b]$$

$$(2.4) \qquad \neg\, b \wedge B \;=\; \neg\, b \wedge B[\mathit{false}/b]$$

Proof: Take separately the cases b is true and b is false. All four results then follow by lemma 1.

□

3 Programs as Relations

Any sequential program possesses a set of variables which characterises its observable state-space. The relation between a program's observable before-states and corresponding after-states provides a simple but useful abstraction of that program's behaviour. A binary relation is defined as a pair (α, P) where

$$\alpha \quad = \quad \alpha_{in} \;\cup\; \alpha'_{out}$$

α_{in} being a set of undashed input variables and α'_{out} a set of dashed output variables, and P is a predicate containing no free variables other than those in

α. Where the context unambiguously determines its alphabet we will often refer to a relation by its predicate alone.

In a *homogeneous* binary relation the output alphabet α'_{out} is obtained from the input alphabet α_{in} simply by putting a dash on each of the latter's variables. In most programming languages the same variables are observable at the beginning and end of the program's execution. A program written in such a language is therefore modelled by a homogeneous binary relation. In this paper we confine our attention from now on to such relations.

A special case of a homogeneous binary relation is one whose predicate contains no free variables other than those in α_{in}. We call such a relation a *condition*.

3.1 Correctness

A major incentive for adoption of the relational model for program descriptions is that it provides an extremely simple representation in terms of our implication ordering, of the notion of correctness of one program description with respect to another. Now suppose we have two program descriptions represented by relations (α, A) and (α, B) over the same alphabet α. Then we say (α, B) is *correct* with respect to (α, A) if $A \sqsubseteq B$. This means that any set of observations which satisfies (α, B) must also satisfy (α, A). Thus any program which implements (α, B) also necessarily implements (α, A).

3.2 Sequential Composition

Another benefit of the relational model is that the sequential composition of programs is simply represented by relational composition. Let (α, A) and (α, B) be homogeneous binary relations whose common alphabet α comprises a list of undashed variables v and a corresponding list of dashed variables v'. We define the predicate $A \,;\, B$ by

Definition 3 (Sequential composition)

$$A \,;\, B \quad = \quad \exists\, v'' \bullet A[v''/v'] \wedge B[v''/v]$$

where v'' is a fresh list of double-dashed variables corresponding with those of v. The sequential composition of relations (α, A) and (α, B) is the relation with alphabet α and predicate $A \,;\, B$.

3.3 Non-deterministic Choice

The non-deterministic choice $[]$ between two programs sharing the same variable-space is also very simply represented in the relational model by logical disjunction. Thus we have

Definition 4.1 (Bounded non-deterministic choice)

$$(\alpha, A) \,[]\, (\alpha, B) \quad =_{df} \quad (\alpha, A \vee B)$$

The concept of non-deterministic choice also generalises to the *unbounded* case. Let A_i be an arbitrary family of predicates, indexed by i which ranges over an indexing set I. Then we have

Definition 4.2 (Unbounded non-deterministic choice)

$$[]_{i \in I}\ (\alpha, A_i) \quad =_{df} \quad (\alpha\, , \exists\, i : I \bullet A_i)$$

3.4 Non-termination

As it stands the relational abstraction has a serious limitation: it is inadequate to reflect a very important aspect of computational behaviour which a sequential program may exhibit: namely, non-termination. Various elaborations of the simple relational model have been proposed to deal with non-termination. For example, the program's state-space can be augmented with a special, or improper, element $\perp$ to represent the non-terminating state. The technical objection to such a device is that it undermines the homogeneity of the program's state-space, thus rendering the model very cumbersome to reason about.

Hoare and He adroitly avoid this disadvantage by using instead *auxiliary* variables, which can be introduced into the alphabet to facilitate reasoning about the program, but which cannot be referenced by the text of the program itself. (Alphabet variables other than auxiliary ones we call *regular* variables.) Hoare and He conscript the auxiliary boolean-valued variables ok and ok' into the program's alphabet of variables; they are not directly observable or manipulable by the program itself, but record respectively the observations that execution of the program has commenced and has terminated.

4 Hoare-He Designs

Hoare and He distinguish a subclass of predicates, which can be written in the form $ok \wedge P \ \Rightarrow\ ok' \wedge Q$ where P and Q do not contain ok or ok': that is, they are predicates over the regular variables only. They define a *design* as a relation whose predicate can be expressed in this form, for which they use the special notation $P \vdash Q$, as in

Definition 5 (Design)

$$P \vdash Q \quad =_{df} \quad ok \wedge P \ \Rightarrow\ ok' \wedge Q$$

It can be informally interpreted as

> If the program starts in circumstances satisfying P, it will terminate in a state satisfying Q.

P is called the *assumption* of the design and Q its *commitment*. Designs play a central role in Hoare and He's theory. They are so-called since, being expressible as pairs of predicates, they are amenable to the familiar calculus of refinements

employed by formal development methods such as B [1], VDM [9], Z [13] and the Refinement Calculus [10, 2]. A significant achievement of Hoare and He's theory is its elucidation of the two-predicate refinement theories of these methods in terms of its own simpler notion of correctness based on the implication ordering between its single-predicate representations of the same designs.

Unfortunately, the form $P \vdash Q$ is not canonical for designs, since it contains some semantic redundancy: we can replace its commitment Q by any predicate Q^* which satisfies $P \Rightarrow Q \sqsubseteq Q^* \sqsubseteq P \wedge Q$; the resulting design $P \vdash Q^*$ remains semantically equivalent to $P \vdash Q$. In particular, $P \vdash P \wedge Q$ and $P \vdash P \Rightarrow Q$ are both equivalent to $P \vdash Q$.

4.1 An Important Design

Our first design is a simple but very important one: if w is a list of all program variables in scope, the design Π, which models the skip program that changes the values of none of those variables, is defined by

Definition 6 (Skip as a design)

$$\Pi \quad =_{df} \quad \text{true} \vdash w' = w$$

4.2 Some Extreme Designs

Of theoretical interest are the designs we can describe over any alphabet, using just the basic predicates true and false:

name	as a design	as a predicate
free assignment	true $\vdash$ true	$ok \Rightarrow ok'$
$abort_{\mathrm{d}}$	false $\vdash$ true	true
	false $\vdash$ false	
magic	true $\vdash$ false	$\neg\, ok$

The design *free assignment* always terminates, assigning unpredictable values to its program variables. The design $abort_{\mathrm{d}}$ may or may not terminate, but if it does it too will assign unpredictable values to its program variables. The d subscript, reminding us $abort_{\mathrm{d}}$ is a design, distinguishes it from another abort we will encounter in section 6.5. The design *magic* is always miraculous, and therefore impossible to implement.

4.3 Healthiness Conditions for Designs

We have seen that a program is modelled in Hoare and He's theory by a relation on an alphabet of unprimed and primed program variables augmented

with the auxiliary variables ok and ok'. The converse question naturally arises: does every such relation model some program? In fact, the answer is no. Hoare and He provide the following four healthiness conditions ([7] definition 3.2.1) which semantically characterise the predicates of those relations which do model implementable program descriptions:

H1	A	$=$	$(ok \Rightarrow A)$
H2	$A[true/ok']$	$\sqsubseteq$	$A[false/ok']$
H3	$A \,;\, \Pi$	$=$	A
H4	$A \,;\, \text{true}$	$=$	true

H1 requires the predicate A makes no stipulation about the initial or final values of the program variables until the program has started. H2 insists that the predicate A is monotonic in the variable ok', in respect of the boolean lattice {true, false} under the implication ordering true $\sqsubseteq$ false. The semantic significance of H2 is that if the predicate A allows failure to terminate in certain circumstances, then in the same circumstances A must also allow termination. Thus in no circumstances can A ever actually *demand* non-termination. Hoare and He have a crucial theorem, [7] theorem 3.2.3 (Healthiness of designs), which states that a relation (α, A) is a design if and only if its predicate A satisfies H1 and H2.

If in addition A satisfies H3 then (α, A) is a design whose assumption refers only to undashed variables. We call such an assumption the *precondition* of the design. Established formal development methods like B, VDM and the Refinement Calculus all restrict the assumptions of their designs to be preconditions.

H4 enforces Dijkstra's *Law of the Excluded Miracle* [3]: it excludes those predicates that would give rise to relations representing miraculous, i.e. unimplementable, programs.

5 Semantic Limitation of Hoare and He's Model

We have already seen how the correctness of one program with respect to another is reflected in the relational model by the implication ordering on their respective relations. But over the history of program semantics at least three distinct yet related notions of program correctness have emerged. We will briefly survey these, then go on to argue why the one embodied by a Hoare and He design may be increasingly inadequate for many of today's emerging actual computing needs.

5.1 Total Correctness versus General Correctness

In the beginning, there was only *partial correctness*. People were content to reason about what results execution of a program would yield should successful termination occur, while disregarding the slightly awkward question of whether or

not the attainment of that termination was inevitable. When Dijkstra [3] introduced his celebrated *weakest-precondition* (wp) predicate-transformer semantics for programs in 1976, he established a semantics of *total correctness*, in which we reason only about results execution of a program must inevitably yield because termination is guaranteed. He endowed his famous guarded command language in [3] with such a semantics. He also proposed a *weakest-liberal-precondition* (wlp) predicate transformer in [3] which facilitates partial-correctness reasoning about programs, but he made no use it there in defining the semantics of his language.

Interestingly, though, by the time [4] appeared fourteen years later it is clear Dijkstra had arrived at the view that a semantics which fully characterises the behaviour of sequential programs must embrace both total correctness *and* partial correctness. That is, we need to be able to reason separately about

1. the results a program might yield, should termination occur, and
2. whether or not such termination is certain to occur.

Thus Dijkstra and Scholten [4] give both a wp and a wlp interpretation of every construct in the guarded command language. Similarly, Nelson [12] employs both wp and wlp in his seminal generalisation of Dijkstra's calculus to admit abstract programs with miraculous behaviour and unbounded non-determinism. Nowadays the term *general correctness*, coined by Jacobs and Gries [8], denotes the semantics which combines both partial and total correctness.

5.2 Tyranny of H2

The effect of Hoare and He's healthiness condition H2 is plain: it ensures that only designs characterised entirely by total correctness are admitted in their theory. But in comparison with its three companion healthiness conditions H1, H2 and H4 it seems somewhat contrived and fails to inspire the same immediate intuitive conviction as do they. We have seen that H2 insists that non-termination should never be required, but in every programming language we can write executable programs which under certain conditions will definitely not terminate. Why then, we might reasonably ask, should we be bound by such a healthiness condition which forbids us from accurately reflecting such behaviour in our relational model?

The answer, as we have already noted, is that Hoare and He require H2 with H1 in [7] theorem 3.2.3 to provide an effective semantic characterisation of a design. H2, then, is the price we must pay for the simple notion of design which Hoare and He are offering us.

5.3 The Interactive Era

As long as we believe there is no useful purpose in ever insisting of a program that in certain circumstances it *must not* terminate, then H2 may be a price we are content to pay as willing subjects of a regime of total correctness. But

we would do well to consider whether such a belief, forged as it doubtless was in an earlier era of sequential batch computing, remains tenable in our present ever more communicational one. Milner has said that today computing *is* interaction[1]. This has important implications for software developers: no longer are we always exclusively concerned with the result of executing our sequential programs through to successful termination. The program we are developing may well be one individual thread which must interact with others in a complex distributed computing system, and we may thus require it to behave reliably in situations where termination is not even in prospect.

A channel-listener, for example, must idle until it detects an incoming message, then deal effectively with such a message if and when it arrives. But since there is never a guarantee that such a message will ever arrive, our channel-listener must be capable in principle of idling indefinitely. There are circumstances, therefore, where non-termination *is* precisely what we demand. To quote Hehner and Gravell [6]:

> With the addition of communication, non-terminating executions can perform useful computation, so a semantics that does not insist on termination is useful.

We would go further and say we need a semantics in which we can insist on non-termination in certain conditions. In short, we must learn to work in a context of general correctness.

6 Recasting the Model

Of course, we seek to retain the central feature of Hoare and He's model that programs are described by single relations, but total correctness must give way to general correctness. We have to extract more meaning out of each relation which describes a program. We start by renouncing our adherence to H2.

6.1 Prescriptions

Since we repudiate H2, Hoare and He's [7] theorem 3.2.3 dictates we must also discard their notion of a design. In its place we propose something new. We distinguish a new family of predicates which can be written in the form $(ok \wedge P \Rightarrow ok') \wedge (ok' \Rightarrow Q \wedge ok)$, where P and Q are subsidiary predicates which do not contain ok or ok'. We call a relation whose predicate can be expressed in this form a *prescription*[2]. We adopt the special notation $P \Vdash Q$ for such a predicate. Thus we have

Definition 7 (Prescription)

$$P \Vdash Q \quad =_{df} \quad (ok \wedge P \Rightarrow ok') \wedge (ok' \Rightarrow Q \wedge ok)$$

[1] Robin Milner, FACS 21^{st} anniversary symposium address, The Royal Society, London, 2 December 1998.

[2] Our use of the term here is distinct from Morris's original use of it in [11].

It can be informally interpreted as

> If the program starts in circumstances satisfying P, it must terminate. Moreover, if it terminates Q will be satisfied, and it must have started, whether or not in circumstances satisfying P.

We echo the terminology for designs by calling P the *assumption* and Q the *commitment* of $P \Vdash Q$. Like that of $P \vdash Q$, termination of $P \Vdash Q$ is guaranteed under its assumption P. The crucial distinction is that $P \Vdash Q$ always guarantees Q on termination, even if that termination occurs fortuitously, whereas $P \vdash Q$ only guarantees Q where termination itself is guaranteed by the assumption P.

6.2 A Standard Alphabet

From now on we assume our relations are homogeneous over an alphabet which includes ok and ok'. The alphabet of such a relation is therefore in the form $\{w, w', ok, ok'\}$, where w represents a list of undashed program variables and w' represents a corresponding list of dashed program variables. We will call such an alphabet a *standard* one, and we will invariably denote it by α.

6.3 An Important Prescription

We define the prescription *skip*, which models the skip program that changes the values of none of the program variables, by

Definition 8 (Skip as a prescription)

$$skip \quad =_{df} \quad \text{true} \Vdash w' = w$$

Thus *skip* has the same operational interpretation as the design Π, but note that

$$skip \quad = \quad (ok \Leftrightarrow ok') \ \wedge \ (ok \Rightarrow w' = w)$$

while

$$\Pi \quad = \quad (ok \Rightarrow ok') \ \wedge \ (ok \Rightarrow w' = w)$$

6.4 Canonical Form

Given prescriptions $P_1 \Vdash Q_1$ and $P_2 \Vdash Q_2$ over the same alphabet, it is easy to show that

$$P_1 \Vdash Q_1 \quad \sqsubseteq \quad P_2 \Vdash Q_2 \quad \text{if and only if} \quad P_2 \sqsubseteq P_1 \ \text{ and } \ Q_1 \sqsubseteq Q_2$$

We note the pleasing orthogonality of the two conditions on the right-hand side, which is certainly not enjoyed by the corresponding rule for designs in [7]

theorem 3.1.2. Furthermore, from the above directly comes the following rule for equality between prescriptions:

$$P_1 \Vdash Q_1 \;=\; P_2 \Vdash Q_2 \quad \text{if and only if} \quad P_1 \;=\; P_2 \;\text{ and }\; Q_1 \;=\; Q_2$$

Thus, in contrast to the design $P \vdash Q$, our $P \Vdash Q$ is indeed a canonical form for prescriptions. The following lemma tells us how to extract the assumption and commitment of a relation we already know to be a prescription:

Lemma 3 (Extract assumption and commitment)

Let (α, A) be a prescription. Then

$$A \;=\; \neg\, A[\text{true}, \text{false}/ok, ok'] \;\Vdash\; A[\text{true}, \text{true}/ok, ok']$$

Proof: use definition 7 to expand the right-hand side, then use lemma 1 (Equality by cases) to split the equality into the four separate cases given by ok, ok' respectively as (true,true), (true,false), (false,true) and (false, false). The four cases are each trivially true.

6.5 Some Extreme Prescriptions

The prescriptions we can describe over any alphabet, using just the basic predicates true and false, are of considerable theoretical significance. We list them in the following table, in which for interest we also quote the equivalent names from Nelson [12]:

name	as a prescription	as a predicate	Nelson's name
free assignment	true $\Vdash$ true	$ok \Leftrightarrow ok'$	*Havoc*
anarchy	false $\Vdash$ true	$ok' \Rightarrow ok$	*Loop* [] *Havoc*
magic	true $\Vdash$ false	$\neg\, ok \wedge \neg\, ok'$	*Fail*
abort	false $\Vdash$ false	$\neg\, ok'$	*Loop*

The prescription *free assignment* behaves like its design namesake: it always terminates, assigning unpredictable values to its program variables. The prescription *anarchy* behaves exactly like the design $abort_{\mathrm{d}}$: it may or may not terminate, but if it does it too assigns unpredictable values to its program variables. The prescription *magic* behaves like its design namesake: it always acts miraculously, so cannot be implemented.

The prescription *abort* has no exact counterpart in the world of designs: in particular, its behaviour is distinct from that of its near namesake, the design $abort_{\mathrm{d}}$. Its assumption false is unsatisfiable so it can never be guaranteed to terminate, and its commitment false is unfulfillable so if it ever did happen to terminate it would achieve the impossible. Therefore, unlike the design $abort_{\mathrm{d}}$ which might terminate with any result, the prescription *abort* precisely models an infinite loop program which never terminates.

6.6 Normal Prescriptions

In practice, we meet mainly prescriptions whose assumptions are preconditions, referring to unprimed program variables only. We describe a prescription with such an assumption as a *normal* prescription.

A normal prescription whose precondition is also subsumed by its commitment, that is, one that can be presented in the form $P \vdash P \wedge Q$, has an interesting operational interpretation: it describes a program which when started in a state satisfying its precondition P will terminate so that Q holds, but when started otherwise will definitely never terminate. In particular, the normal prescription $P \vdash P \wedge w' = w$ terminates with no changes to the program variables w when started in a state statisfying P, but fails to terminate otherwise. It corresponds to what is historically known as a Floyd assertion [5]. On the other hand, $\text{true} \vdash P \wedge w' = w$ corresponds to a Floyd assumption.

6.7 Healthiness of prescriptions

So far we have only characterised our prescriptions syntactically. We wish to emulate Hoare and He's [7] theorem 3.2.3, by formulating a healthiness condition which semantically characterises a prescription. We propose a new healthiness condition we will call HP, formally expressed as

$$\text{HP} \qquad A[\mathit{false}/\mathit{ok}] \quad = \quad \neg\, \mathit{ok}'$$

The semantic significance of HP is that all that can be asserted about a program whose execution hasn't started is that it hasn't finished. It seems to do no more than capture an intuitively incontestible basic principle of computation, so innocuous that one may be slightly surprised to perceive that Hoare and He's designs contravene it. That HP does in fact precisely characterise a prescription is assured by the following theorem:

Theorem 1 (Healthiness of prescriptions)

A relation (α, A) is a prescription if and only if its predicate A satisfies HP.

Proof of $\Rightarrow$

Let (A, α) be a prescription. Then $A = P \vdash Q$ for some predicates P and Q not involving ok or ok', and by definition of $P \vdash Q$ we have

$A \quad = \quad (ok \wedge P \;\Rightarrow\; ok') \;\wedge\; (ok' \;\Rightarrow\; Q \wedge ok)$

hence by substitution of false for ok

$A[\text{false}/ok] \quad = \quad (\text{false} \wedge P \;\Rightarrow\; ok') \;\wedge\; (ok' \;\Rightarrow\; Q \wedge \text{false})$

simplifying to

$A[\text{false}/ok] \quad = \quad \text{true} \wedge \neg\, ok' \quad = \quad \neg\, ok'$

Proof of $\Leftarrow$

We introduce the following abbreviations for convenience:

$$\begin{array}{lcl} A_t & = & A[\text{true}/ok] \\ A_f & = & A[\text{false}/ok] \\ A_{t,t} & = & A[\text{true},\text{true}/ok, ok'] \\ A_{t,f} & = & A[\text{true},\text{false}/ok, ok'] \end{array}$$

We will prove that

$A \quad = \quad \neg\, A_{t,f} \Vdash A_{t,t}$

$\equiv \quad \{\text{defn of } \neg\, A_{t,f} \Vdash A_{t,t}\}$

$A \quad = \quad (ok \wedge \neg\, A_{t,f} \;\Rightarrow\; ok') \;\wedge\; (ok' \;\Rightarrow\; A_{t,t} \wedge ok)$

$\equiv \quad \{\text{predicate logic}\}$

$A \quad = \quad (ok \;\Rightarrow\; A_{t,f} \vee ok') \;\wedge\; (ok' \;\Rightarrow\; A_{t,t} \wedge ok)$

We know from lemma 1 we can do this by taking the cases ok is true and false separately. The first case we have to prove, with ok true, is

$A_t \quad = \quad (A_{t,f} \vee ok') \;\wedge\; (ok' \Rightarrow A_{t,t})$

$\equiv \quad \{\text{predicate logic}\}$

$A_t \quad = \quad (\neg\, ok' \Rightarrow A_{t,f}) \;\wedge\; (ok' \;\Rightarrow\; A_{t,t})$

which holds by lemma 2.1.

The second case we have to prove, with ok false, is

$A_f \quad = \quad (\text{false} \;\Rightarrow\; A_{t,f} \vee ok') \;\wedge\; (ok' \;\Rightarrow\; A_{t,t} \wedge \text{false})$

$\equiv \quad \{\text{predicate logic}\}$

$A_f \quad = \quad \neg\, ok$

which holds according to our hypothesis HP.

□

From theorem 1 we can derive the following important corollary:

Corollary 1.1 (Closure of prescriptions)

Within the class of all relations over a standard alphabet α, the subclass of prescriptions is closed under sequential composition and non-deterministic choice.

Proof

Let (α, A) and (α, B) be prescriptions.

$A \,;\, B \quad = \quad \{\text{defn of } ;\}$

$\exists\, w'', ok'' \;\bullet\; A[w'', ok''/w', ok'] \;\wedge\; B[w'', ok''/w, ok]$

hence $\{\text{substituting false for ok}\}$

$(A \,;\, B)[\mathit{false}/ok] \quad =$

$\exists\, w'', ok'' \;\bullet\; A[\mathit{false}, w'', ok''/ok, w', ok'] \;\wedge\; B[w'', ok''/w, ok]$

$=$ {as a prescription A satisfies HP}

$\exists\, w'', ok'' \bullet \neg\, ok' \ \wedge\ B[w'', ok''/w, ok]$

$=$ {change names of bound variables}

$\exists\, w, ok \bullet \neg\, ok' \ \wedge\ B$

$=$ {take $\neg\, ok'$ outside $\exists$}

$\neg\, ok' \ \wedge\ \exists\, w, ok \bullet B$

$=$ {express $\exists\, ok$ as disjunction of cases true and false}

$\neg\, ok' \ \wedge\ (\exists\, w \bullet B[true/ok] \vee B[false/ok])$

$=$ {since as a prescription B satisfies HP by theorem 1}

$\neg\, ok' \ \wedge\ (\exists\, w \bullet B[true/ok] \vee \neg\, ok')$

$=$ {predicate logic}

$\neg\, ok'$ So $A\ ;\ B$ satisfies HP.

Now let (α, A_i) be an arbitrary family of prescriptions. Then, by definition of non-determinstic choice, $[]_{i \in I}\ (\alpha, A_i) \quad = \quad (\alpha\ , \exists\, i \bullet A_i)$.

We have $(\exists\, i \bullet A_i)[false/ok]$

$=$ {substitution distributes through $\exists$}

$\exists\, i \bullet A_i[false/ok]$

$=$ {each A_i satisfies HP}

$\exists\, i \bullet \neg\, ok'$

$=$ {discard superfluous quantification}

$\neg\, ok'$ So $\exists\, i \bullet A_i$ satisfies HP.

□

It is also easy to show that normal prescriptions are closed under arbitrary non-deterministic choice. In particular, it follows directly from the definition of $[]$ that

$$(P_1 \Vdash Q_1)\ []\ (P_2 \Vdash Q_2) \quad = \quad (P_1 \wedge P_2) \Vdash (Q_1 \vee Q_2)$$

6.8 The Lattice Of Prescriptions

The prescriptions over a standard alphabet α comprise a full sub-lattice of the lattice of all relations over that alphabet under the implication ordering $\sqsubseteq$. The bottom of the sub-lattice is *anarchy* and the top is *magic*. In fact we have the following alternative characterisation theorem for prescriptions:

Theorem 2 (Ordering of prescriptions)

A relation (α, A) is a prescription if and only if its predicate A satisfies

$$anarchy \ \sqsubseteq\ A \ \sqsubseteq\ magic$$

Proof of $\Rightarrow$

Assume (α, A) is a prescription. Then by theorem 1
A satisfies HP, so
$A[\text{false}/ok] \;=\; \neg\, ok'$
Now by lemma 2.1 for any predicate A
$A \quad = \quad (ok \Rightarrow A[\text{true}/ok]) \;\wedge\; (\neg\, ok \Rightarrow A[\text{false}/ok])$
and hence
$\neg\, ok \Rightarrow A[\text{false}/ok] \;\sqsubseteq\; A$
and hence, since A satisfies HP,
$\neg\, ok \Rightarrow \neg\, ok' \;\sqsubseteq\; A$
or, equivalently,
$ok' \Rightarrow ok \;\sqsubseteq\; A$

Also by lemma 2.2 we have
$A \quad = \quad (ok \wedge A[\text{true}/ok]) \;\vee\; (\neg\, ok \wedge A[\text{false}/ok])$
and hence
$A \;\sqsubseteq\; \neg\, ok \wedge A[\text{false}/ok]$
and hence, since A satisfies HP,
$A \;\sqsubseteq\; \neg\, ok \wedge \neg\, ok'$

Proof of $\Leftarrow$

Assume A satisfies
$ok' \Rightarrow ok \;\sqsubseteq\; A \;\sqsubseteq\; \neg\, ok \wedge \neg\, ok'$
Then by substituting false for ok we have
$ok' \Rightarrow \text{false} \;\sqsubseteq\; A[\text{false}/ok] \;\sqsubseteq\; \text{true} \wedge \neg\, ok'$
or, equivalently,
$\neg\, ok' \;\sqsubseteq\; A[\text{false}/ok] \;\sqsubseteq\; \neg\, ok'$
or, equivalently,
$A[\text{false}/ok] \quad = \quad \neg\, ok'$

□

6.9 A Lemma about *skip*

It is convenient here to prove the following technical lemma about *skip*, which will be needed in the next section:

Lemma 4 (Left multiplier effect of *skip*)

Let R be any relation (not necessarily a prescription) over a standard alphabet

α. Then

$$skip \ ; R \quad = \quad (\exists\, w \ \bullet \ \neg\ ok \wedge R[false/ok]) \ \vee \ R$$

Proof

$skip \ ; R$
$=$ {defn of $skip$}
$((ok \Leftrightarrow ok') \ \wedge \ (ok' \Rightarrow w' = w)) \ ; R$
$=$ {defn of ;}
$\exists\, w'', ok'' \ \bullet \ (ok \Leftrightarrow ok'') \ \wedge \ (ok'' \Rightarrow w'' = w) \ \wedge \ R[w'', ok''/w, ok]$
$=$ {one-point rule for ok''}
$\exists\, w'' \ \bullet \ (ok \Rightarrow w'' = w) \ \wedge \ R[w''/w]$
$=$ {predicate logic}
$(\exists\, w'' \ \bullet \ \neg\ ok \wedge R[w''/w]) \ \vee \ \exists\, w'' \ \bullet \ w'' = w \ \wedge \ R[w''/w]$
$=$ {change name of bound variable, and one-point rule for w''}
$(\exists\, w \ \bullet \ \neg\ ok \wedge R) \ \vee \ R$
$=$ {lemma 2.4}
$(\exists\, w \ \bullet \ \neg\ ok \wedge R[false/ok]) \ \vee \ R$

□

7 Algebraic Characterisation of Prescriptions

In the introduction we alluded to a class of relations which could be characterised algebraically, entirely by means of multiplicative laws like the zero and unit laws for sequential composition. We now show how our prescriptions comprise just such a class.

7.1 The Special Roles of *abort* and *skip*

Within the class of all prescriptions over an alphabet, the prescription *abort* has a special significance: it is a left zero of sequential composition. Similarly, the prescription *skip* has a distinguished role as a left unit of sequential composition. Indeed, more than this, we can actually characterise prescriptions precisely in terms of such a left zero law and left unit law. We have

Theorem 3 (Left unit and left zero for prescriptions)

A relation (α, A), over a standard alphabet α, is a prescription if and only if

$$skip \ ; A \ = \ A \qquad \text{and} \qquad abort \ ; A \ = \ abort$$

Proof of $\Rightarrow$

Let (A, α) be a prescription over a standard alphabet α. Then $A = P \Vdash Q$ for some predicates P and Q not involving ok or ok'. So we have

$abort \ ; A$

$=$ {defn of $abort$ and defn of $A = P \Vdash Q$}

$\neg\, ok' \ ; \ ((ok \wedge P \ \Rightarrow \ ok') \ \wedge \ (ok' \ \Rightarrow \ Q \wedge ok))$

$=$ {defn of ;}

$\exists\, w'', ok'' \bullet \neg\, ok'' \ \wedge \ (ok'' \wedge P[w''/w] \ \Rightarrow \ ok') \ \wedge \ (ok' \ \Rightarrow \ Q[w''/w] \wedge ok'')$

$=$ {one-point rule for ok''}

$\exists\, w'' \bullet \ ok' \ \Rightarrow \ Q[w''/w] \wedge \text{false}$

$= \quad \neg\, ok' \quad = \quad abort$

Similarly

$skip \ ; A$

$=$ {lemma 4}

$(\exists\, w \bullet \neg\, ok \wedge A[false/ok]) \ \vee \ A$

$=$ {A is a prescription so by theorem 1}

$(\exists\, w \bullet \neg\, ok \wedge \neg\, ok') \ \vee \ A$

$=$ {discard superfluous quantification}

$(\neg\, ok \wedge \neg\, ok') \ \vee \ A$

$=$ {defn of $magic$}

$magic \ \vee \ A$

$=$ {$A \sqsubseteq magic$ by theorem 2}

A

Proof of $\Leftarrow$

Assume A is a predicate such that $abort \ ; A = abort$ and $skip \ ; A = A$. Then we have

$abort \ ; A \ = \ abort$

$\equiv$ {defns of $abort$ and ;}

$\exists\, w'', ok'' \bullet \neg\, ok'' \ \wedge \ A[w'', ok''/w, ok] \ = \ \neg\, ok'$

$\equiv$ {one-point rule for ok''}

$\exists\, w'' \bullet A[w'', false/w, ok] \ = \ \neg\, ok'$

$\equiv$ {change of bound variable}

$\exists\, w \bullet A[false/ok] \ = \ \neg\, ok'$ {Result 1}

Also, we have

$skip \ ; A \ = \ A$

$\equiv$ {lemma 4}

$(\exists\, w \bullet \neg\, ok \wedge A[false/ok]) \;\vee\; A \;=\; A$

hence {specific case of ok false}

$(\exists\, w \bullet A[false/ok]) \;\vee\; A[false/ok] \;=\; A[false/ok]$

$\equiv$ {predicate logic}

$A[false/ok] \;\sqsubseteq\; (\exists\, w \bullet A[false/ok])$

$\equiv$ {for any x, p we have $(\exists\, x \bullet p) \sqsubseteq p$}

$A[false/ok] \;=\; \exists\, w \bullet A[false/ok]$

$\equiv$ {Result 1 above}

$A[false/ok] \;=\; \neg\, ok'$

So A satisfies HP, and thus by theorem 1 A is a prescription.

□

7.2 Normality

Remember, a *normal* prescription is one whose assumption is independent of any primed variable. We can characterise a normal prescription algebraically, as shown by the following theorem, which is our counterpart for prescriptions of Hoare and He's [7] theorem 3.2.4 for designs:

Theorem 4 (Normal prescription)

A prescription (α, A) over a standard alphabet α, is a normal prescription if and only if $A\,;\, skip \;=\; A$.

Proof

$A\,;\, skip \;=\; A$

$\equiv$ {defn of $skip$}

$A \;;\; (ok \Leftrightarrow ok') \;\wedge\; (ok' \Rightarrow w' = w) \;=\; A$

$\equiv$ {predicate logic}

$A \;;\; ((ok \Leftrightarrow ok') \wedge \neg\, ok') \;\vee\; ((ok \Leftrightarrow ok') \wedge w' = w)) \;=\; A$

$\equiv$ {; distributes over $\vee$}

$A\,;\, ((ok \Leftrightarrow ok') \wedge \neg\, ok') \;\vee\; A\,;\, ((ok \Leftrightarrow ok') \wedge w' = w)) \;=\; A$

$\equiv$ {defn of ;}

$(\exists\, w' \bullet A[\text{false}/ok'] \wedge \neg\, ok') \;\vee\; A \;=\; A$

$\equiv$ {predicate logic}

$A \;\sqsubseteq\; \exists\, w' \bullet A[\text{false}/ok'] \wedge \neg\, ok'$

$\equiv$ {case ok' is false (other case is trivial)}

$A[\text{false}/ok'] \;\sqsubseteq\; \exists\, w' \bullet A[\text{false}/ok']$

$\equiv$ {predicate logic}

$A[\text{false}/ok']$ is independent of w'

$\equiv$ {$A = P \Vdash Q$ for some P and Q, so $A[\text{false}/ok'] = \neg (P \wedge ok)$}

P, the assumption of A, is independent of w'.

□

7.3 Feasibility

A prescription is said to be *miraculous* if it ever guarantees termination in an impossible state. Such a prescription would, of course, be unimplementable. Otherwise a prescription is said to be *feasible*. We have an algebraic characterisation of a feasible prescription, expressed in the following theorem, which is our counterpart for prescriptions of Hoare and He's [7] theorem 3.2.5 for designs:

Theorem 5 (Feasible prescription)

A prescription (α, A) over a standard alphabet α, is feasible if and only if $A \,;\, abort = abort$.

Proof

$A \,;\, abort = abort$

$\equiv$ {defn of $abort$}

$A \,;\, \neg\, ok' = \neg\, ok'$

$\equiv$ {defn of ;}

$\exists\, w'', ok'' \bullet A[w'', ok''/w', ok'] \wedge \neg\, ok' = \neg\, ok'$

$\equiv$ {predicate logic}

$\exists\, w'', ok'' \bullet A[w'', ok''/w', ok'] \sqsubseteq \neg\, ok'$

$\equiv$ {change names of bound variables}

$\exists\, w', ok' \bullet A \sqsubseteq \neg\, ok'$

$\equiv$ {case ok' is false (other case is trivial)}

$\exists\, w', ok' \bullet A \sqsubseteq \text{true}$

$\equiv$ {predicate logic}

$\exists\, w', ok' \bullet A$

□

7.4 Full Characterisation of a Normal Feasible Prescription

Gathering together the results of theorems 3, 4 and 5, we have that a relation (α, A) over a standard alphabet α, is a normal feasible prescription if and only if it satisfies the following laws:

$$abort \,;\, A = abort = A \,;\, abort$$

and

$$skip \ ; A \quad = \quad A \quad = \quad A \ ; skip$$

Such a satisfying result, together with our recognition that such prescriptions more closely model programming reality than designs, impels us forcefully to a recognition that our concept of a prescription is indeed a natural completion of Hoare and He's concept of a design.

8 Sequential Composition

We have already seen by corollary 1.1 that the class of prescriptions over an alphabet is closed under sequential composition. Now we give an explicit form for prescription sequential composition:

Theorem 6 (Sequential composition of prescriptions)

Let $P_1 \Vdash Q_1$ and $P_2 \Vdash Q_2$ be prescriptions over the same standard alphabet α. Then

$$(P_1 \Vdash Q_1) \ ; (P_2 \Vdash Q_2) \quad = \quad (\forall\, w' \bullet P_1) \wedge \neg\, (Q_1 \ ; \neg\, P_2) \ \Vdash \ Q_1 \ ; Q_2$$

Proof

Corollary 1.1 already assures that the left-hand side is a prescription; we use lemma 3 to extract its assumption and commitment. Let A be $P_1 \Vdash Q_1$ and B be $P_2 \Vdash Q_2$. By lemma 3 the commitment of $A \ ; B$ as

$(A \ ; B)[true, true/ok, ok']$

$=$ {defn of ; and absorb substitution}

$\exists\, w'', ok'' \ \bullet \ A[w'', true, ok''/w', ok, ok'] \wedge B[w'', ok'', true/w, ok, ok']$

$=$ {ok'' is true or false}

$\exists\, w'' \ \bullet \ A[w'', true, true/w', ok, ok'] \wedge B[w'', true, true/w, ok, ok']$

$\quad \vee \ A[w'', true, false/w', ok, ok'] \wedge B[w'', false, true/w, ok, ok']$

$=$ {B is a prescription so $B[false, true/ok, ok']$ is false}

$\exists\, w'' \ \bullet \ A[w'', true, true/w', ok, ok'] \wedge B[w'', true, true/w, ok, ok'] \ \vee \ false$

$=$ {$A[true, true/ok, ok'] \ = \ Q_1, B[true, true/ok, ok'] \ = \ Q_2$}

$\exists\, w'' \bullet Q_1[w''/w'] \wedge Q_2[w''/w]$

$=$ {defn of ;}

$Q_1 \ ; Q_2$

Similarly, by lemma 3 the assumption of $A \ ; B$ is

$\neg\, (A; B)[true, false/ok, ok']$

$=$ {defn of ; and absorb substitution}

$$
\begin{array}{ll}
\neg\ \exists\, w'', ok''\ \bullet\ A[w'', true, ok''/w', ok, ok'] \wedge B[w'', ok'', false/w, ok, ok'] \\
= & \{ok'' \text{ is true or false}\} \\
\neg\ \exists\, w''\ \bullet\ A[w'', true, true/w', ok, ok'] \wedge B[w'', true, false/w, ok, ok'] \\
\qquad\qquad \vee\ A[w'', true, false/w', ok, ok'] \wedge B[w'', false, false/w, ok, ok'] \\
= & \{B \text{ is a prescription so } B[false, false/ok, ok'] \text{ is true}\} \\
\neg\ \exists\, w''\ \bullet\ A[w'', true, true/w', ok, ok'] \wedge B[w'', true, false/w, ok, ok'] \\
\qquad\qquad \vee\ A[w'', true, false/w', ok, ok'] \\
= & \{\text{defns of } A \text{ and } B \text{ and lemma 3 on } A \text{ and } B\} \\
\neg\ \exists\, w''\ \bullet\ Q_1[w''/w'] \wedge \neg\ P_2[w''/w]\ \vee\ \neg\ P_1[w''/w'] \\
= & \{\text{distribute } \neg\ \exists \text{ across } \vee\} \\
(\neg\ \exists\, w''\bullet Q_1[w''/w'] \wedge \neg\ P_2[w''/w])\ \wedge\ (\neg\ \exists\, w''\bullet \neg\ P_1[w''/w']) \\
= & \{\text{defn of } ; \text{ and predicate logic}\} \\
\neg\ (Q_1\ ;\neg\ P_2)\ \wedge\ \forall\, w''\bullet P_1[w''/w'] \\
= & \{\text{change name of bound variable}\} \\
\neg\ (Q_1\ ;\neg\ P_2)\ \wedge\ \forall\, w'\bullet P_1
\end{array}
$$

□

We have an immediate corollary of theorem 6 concerning the sequential composition of normal prescriptions:

Corollary 6.1 (Sequential composition of normal prescriptions)

Normal prescriptions are closed under sequential composition. Let $P_1 \Vdash Q_1$ and $P_2 \Vdash Q_2$ be normal prescriptions over the same alphabet. Then

$$(P_1 \Vdash Q_1)\ ;(P_2 \Vdash Q_2) \quad = \quad P_1 \wedge \neg\ (Q_1\ ;\neg\ P_2)\ \Vdash\ Q_1\ ;\ Q_2$$

Proof: by theorem 6, but noting that if P_1 is a precondition then w' is not free in P_1, and so $\forall\, w'\bullet P_1$ reduces to P_1. Also, if P_2 is a precondition, then w' is not free in $(Q_1\ ;\neg\ P_2)$, and so the assumption of the prescription on the right-hand side is a precondition.

□

9 Relaxed Prescriptions

We can *relax* the prescription $P \Vdash Q$ to derive the related prescription

$$P\ \Vdash\ P \Rightarrow Q$$

We will denote by $relax(\_)$ the prescription transformer that maps in this way a prescription A to its corresponding relaxed prescription $relax(A)$. The subclass of relaxed prescriptions is semantically characterised by augmenting our fundamental prescriptive healthiness condition HP with Hoare and He's H2.

Although closed under non-deterministic choice, the class of relaxed prescriptions is not closed under sequential composition. For example, *anarchy* and *magic* are both relaxed prescriptions, yet

$$anarchy \ ; magic \quad = \quad abort$$

which is not a relaxed prescription, since $relax(abort) = anarchy$. We therefore define a another sequential composition operator $;_{rlx}$ by

Definition 9 (Relaxed sequential composition)

$$A \ ;_{rlx} B \quad =_{df} \quad relax(A \ ; B)$$

The class of Hoare and He designs is congruent to the class of relaxed prescriptions under this relaxed sequential composition.

10 Conclusion

We have reviewed Hoare and He's single-predicate model of sequential programs, and uncovered a shortcoming in its ability always to provide an adequate interpretation of the behaviour of a sequential program, such as might be required if that program is employed as a sequential component of an interactive co-operating system. We have demonstrated how the single-predicate model can be recast to overcome this limitation, using a variation on the predicate syntactic form employed by Hoare and He for their designs, which we have called a *prescription*.

We have shown that our prescriptions have a remarkably simple semantic characterisation by means of our single intuitively compelling healthiness condition HP. We have seen that our model also admits a pleasingly simple algebraic formulation which reveals it as a natural completion of Hoare and He's model. We have identified our subclass of relaxed prescriptions, which can be regarded as an endomorphic embedding of Hoare and He's class of designs within the class of prescriptions.

We would maintain our prescriptions afford a finer description of actual sequential program behaviour than do designs, a description for example which embraces certain aspects of behaviour such as guaranteed non-termination, which are deliberately excluded by Hoare and He for designs. Of course, such aspects of behaviour are only relevant in the realm of parallel or reactive computing where individual sequential programs have to interact with their environments. To suggest that there is competition between the two models would be to mislead. Hoare and He's model addresses the realm of pure sequential computing, where no such interactions are allowed, and it is a perfectly adequate model for such a realm. Only when we broaden our attention from this realm to that of reactive computing does our prescriptional model supersede that of Hoare and He.

References

1. J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.
2. R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction.* Springer-Verlag New York, 1998.
3. E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall International, 1976.
4. E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Springer Berlin, 1990.
5. R.W. Floyd. Assigning meanings to programs. *Proceedings of Symposia in Applied Mathematics*, 19:19–32, 1967.
6. E.C.R. Hehner and A.M. Gravell. Refinement semantics and loop rules. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 - Formal Methods*, number 1709 in Lecture Notes in Computer Science, pages 1497–1510. Springer-Verlag, 1999.
7. C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming.* Prentice Hall, 1998.
8. D. Jacobs and D. Gries. General correctness: a unification of partial and total correctness. *Acta Informatica*, 22:67–83, 1985.
9. C.B. Jones. *Systematic Software Development Using VDM (2nd edn).* Prentice-Hall, 1990.
10. C.C. Morgan. *Programming from Specifications (2nd edn).* Prentice Hall International, 1994.
11. J.M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer programming*, 9:287–306, 1987.
12. G. Nelson. A generalisation of Dijkstra's calculus. *ACM Transactions on Programmg Languages and Systems*, 11(4), 1989.
13. J.M. Spivey. *The Z Notation: a Reference Manual (2nd edn).* Prentice Hall, 1992.