Electronic Notes in Theoretical Computer Science 137 (2005) 45–66 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Angelic Nondeterminism and Unifying Theories of Programming

Ana Cavalcanti [a](#_bookmark0),[1](#_bookmark1) ,[2](#_bookmark1) Jim Woodcock[a](#_bookmark0),[1](#_bookmark1) ,[3](#_bookmark1)

a *Department of Computer Science University of York*

*York, England*

Abstract

Hoare and He’s unifying theories of programming (UTP) is a model of alphabetised relations expressed as predicates, which supports development in several programming paradigms. The aim is the unification of languages and techniques, so that we can benefit from results in different contexts. In this paper, we investigate the integration of angelic nondeterminism in the UTP; we propose the unification of a model of binary multirelations, which is isomorphic to the monotonic predicate transformers model and can express angelic and demonic nondeterminism.

*Keywords:* semantics, refinement, relations, predicate transformers.

# Introduction

Angelic nondeterminism is a specification and programming concept that is typically available in unified languages of refinement calculi [[18](#_bookmark26),[4](#_bookmark14)], and in concurrent constraint programming languages [[15](#_bookmark27)]. In program development techniques, it is reflected in choice constructs in which the choice is not arbi- trary, but made to guarantee success, if possible. In programming languages, it is reflected in the use of backtracking. The work in [[16](#_bookmark28)] explores angelic nondeterminism in a language for definition of tactics of proofs.

1 The authors are grateful to Will Harwood for extensive discussions, and to Carroll Morgan for pointing out the work on binary multirelations.

2 Email:Ana.Cavalcanti at cs.york.ac.uk

3 Email: jim at cs.york.ac.uk

1571-0661 © 2005 Elsevier B.V. Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2005.04.024

In contrast, demonic nondeterminism is related to an arbitrary choice con- struct that provides no guarantees; success is still a possibility, but it does not influence the choice. Demonic choice is commonly used to model abstrac- tion and information hiding; in this case, choice is used in a specification to explicitly indicate options that are left open to the programmer.

In [[11](#_bookmark22)], Gardiner and Morgan identify angelic choice with the least upper bound in the lattice of monotonic predicate transformers. In [[19](#_bookmark29)], they use this construct to define logical constants, which are pervasive in refinement tech- niques, and are sometimes named logical, auxiliary, or angelic variables. Log- ical constants play a fundamental roˆle in the formalisation of data refinement of recursive programs, and, more importantly, they are used in calculational simulation rules for specification statements and guarded commands.

In [[18](#_bookmark26)] Morgan proposes an algebraic approach to refinement. In that work, logical constants are at the heart of the formalisation of initial variables, which are used in specification statements: they appear in postconditions to refer to values of variables before the execution of the program. Logical constants are also central to the stepwise calculational development of sequences and loops. Back and von Wright’s work on refinement [[4](#_bookmark14)] has also explored the use of angelic nondeterminism. They haves studied the set of monotonic predicate transformers as a lattice with the refinement ordering. They have identi- fied interesting sublattices, in which choice can be either angelic or demonic, and a complete base language, which can describe any monotonic predicate transformer [[1](#_bookmark12),[2](#_bookmark13)]. More recently, they have suggested the use of angelic nonde- terminism to model user interactions with a system, and game-like situations. Morgan’s refinement calculus has been adapted to handle Z specifications; the resulting calculus is called ZRC [[7](#_bookmark18)]. It is incorporated in *Circus* [[21](#_bookmark32)], a combination of Z and CSP that supports refinement of state-rich, reactive programs. The design of *Circus* follows the trend to combine notations; it has been successfully applied in case studies, and has a refinement technique that supports decomposition of the state and behaviour of centralised systems [[5](#_bookmark16)]. Departing from standard work in refinement calculi, the semantic model of *Circus* is based on the unifying theories of programming (UTP) [[14](#_bookmark25),[22](#_bookmark33)] of Hoare and He. This is a predicate-based relational model that links constructs in several programming paradigms: imperative, concurrent, logical, and others. By providing a framework for the study of state and reactive aspects of a program, the UTP has proved to be very adequate as a basis for the *Circus* model, and for several of its extensions. Nevertheless, logical constants and, more generally, angelic nondeterminism are not considered. Since we adopt Morgan’s calculational refinement style, we have pursued the possibility of

modelling angelic nondeterminism in the UTP.

Angelic nondeterminism has been extensively studied using weakest pre- condition semantics. There are results on the relationship between relational and predicate transformer models in which relations are sets of pairs of states and predicates are sets of states [[12](#_bookmark23),[6](#_bookmark17)]. These results establish that the UTP relational model cannot capture angelic and demonic nondeterminism.

In this paper, firstly, we consider a set-based relational model for the UTP. Secondly, we propose a predicate transformer model; conjunctive predicate transformers correspond to the set-based relations, and therefore to UTP re- lations. These models clarify some aspects of the UTP, and provide guidance on the use of the model of binary multirelations introduced in [[20](#_bookmark30)] for the UTP. Based on this model, we propose a UTP theory for angelic nondeterminism.

In the next section, we present an overview of the unifying theories of programming. In Section [3](#_bookmark3), we consider a set-based relational model and a predicate transformer model for the UTP. In Section [4](#_bookmark7) we enrich the UTP with a theory to cope with angelic and demonic nondeterminism. Finally, in Section [5](#_bookmark11) we present our conclusions and directions for future work.

# Unifying theories of programming

The objective of Hoare and He’s unifying theories of programming is to study and compare programming paradigms. The main concern is with program development; using the framework of the UTP, it should be possible to take advantage of different techniques and approaches whenever convenient.

In the general theory of relations of the UTP, a relation is a pair (*αP, P* ), where *αP* is a set of names of observational variables, and *P* is a predicate. The set of variables is the alphabet of the relation; it contains both the set *inαP* of undashed names of the observational variables, and the set *outαP* of dashed names. The free variables of *P* must be contained in *αP* .

Each observational variable records information relevant to characterise the behaviour of a program. For example, program variables are observational variables; the model of an assignment *x* := *e*, if the program variables are *x* , *y* , and *z* , is as follows. The undecorated name of a variable refers to its value before the execution of the program, and the dashed name refers to its value in a subsequent observation: final or intermediate.

' ' '

*x* := *e* =^ (*x* = *e* ∧ *y* = *y* ∧ *z* = *z* )

The alphabet is { *x , y, z , x* '*, y* '*, z* ' }. The assignment sets the final value of *x* , which is represented by *x* ', to *e*; all the other variables are unchanged.

The program *II* =^ (*v* ' = *v* ) skips: it does not change the observational

variables *v* . We write *v* ' = *v* as an abbreviation for a conjunction of equalities

that state that the final value of each variable is equal to its initial value.

A sequence *P* ; *Q* is defined simply as relational composition, if, for each dashed variable in the alphabet of *P* , the undashed variable is in the alphabet of *Q* . The set *inα*'*Q* is obtained by dashing all variables in *αQ* .

*P* (*v* '); *Q* (*v* ) =^

E *v*0 • *P* (*v*0) Λ *Q* (*v*0) provided *outαP* = *inα*'*Q* = { *v* ' }

The notation *P* (*v* ') emphasises that *P* may have free occurrences of observa- tional variables *v* '; *P* (*v*0) is the predicate obtained by substituting *v*0 for the free occurrences of *v* ' in *P* . Similarly, for *Q* (*v* ) and *Q* (*v*0).

The nondeterministic choice *P* H *Q* =^ *P* V *Q* of relations *P* and *Q* with

the same alphabet is demonic. It behaves like either *P* or *Q* .

The set of relations with a particular alphabet is a complete lattice, with order ⇐; this is the refinement ordering. More formally, the program denoted by *P* is refined by that denoted by *Q* when [*Q* ⇒ *P* ]. As a matter of fact, *P* and *Q* can be either programs (assignments, sequence, choices, and others) or any relation used to specify a program; they are all relations. The square brackets denote universal quantification over all the alphabet.

In contrast with the other operators, the least upper bound H *S* of a set *S*

of relations is defined algebraically: [*P* ⇐ H *S* ] =^ ([*P* ⇐ *X* ] for all *X* in *S* ).

The bottom of this lattice is the program ⊥ =^ true, which is called abort.

Incidentally, the top element is false; it is written T and called miracle.

Recursion is modelled using least fixed points. If *F* (*X* ) is a relation, in which *X* is used as a recursion variable, the recursive program is written *µ X* • *F* (*X* ). This is the least fixed point of the function *F* .

Hoare and He point out what they regard an infelicity. The recursive program *µ X* • *X* is supposed to model an infinite loop; it is equivalent to ⊥ or true. Nevertheless, the sequence (*µ X* • *X* ); *x* ' = 3 is equivalent to *x* ' = 3, even though it should not be possible to recover from non-termination.

The solution proposed by Hoare and He is the introduction of an extra boolean observational variable *ok* to record termination. If *ok* has value *true*, it means that the program has started; if *ok* ' has value *true*, then the program has terminated. In this new theory, relations take the form of designs *P* ▶ *Q* .

(*P* ▶ *Q* ) =^

(*ok* Λ *P* ) ⇒ (*ok* ' Λ *Q* )

The predicates *P* and *Q* are the program’s pre and postcondition. If the design has started and *P* holds, then it terminates and establishes *Q* .

In this new theory, assignment and skip are redefined. Below, *y* and *y* '

|  |  |  |
| --- | --- | --- |
| H1 | *R* = (*ok* ⇒ *R*) | No predictions before startup |
| H2 | [*R*[*false/ok* '] ⇒ *R*[*true/ok* ']] | Non-termination is not required |
| H3 | *R* = *R* ; *II* | Preconditions do not use dashes |
| H4 | *R* ; true = true | Feasibility |

Table 1

UTP Healthiness conditions

stand for the observational variables other than *x* and *x* '.

' '

*x* := *e* =^ true ▶ *x* = *e* Λ *y* = *y*

*II* =^ true ▶ *v* ' = *v*

The new definitions use designs to take *ok* and *ok* ' into account.

Four healthiness conditions on relations *R* are regarded of interest in the theory of designs; they are summarised in Table [1](#_bookmark2). Healthiness condition H1 states that any restrictions on the behaviour of *R* only need to hold if it has started. The second healthiness condition states that *R* cannot require non-termination: if it holds when *ok* ' is *false*, then it also holds when *ok* ' is *true*. Together, H1 and H2 characterise the relations that can be expressed as designs: a predicate is H1 and H2 if and only if it is a design.

The healthiness conditions H3 and H4 are expressed as equations between programming constructs. Results presented in [[14](#_bookmark25)] clarify that H3 designs can be expressed using preconditions that do not refer to dashed observational variables, and that H4 designs model feasible or implementable programs.

Designs form a UTP theory that is characterised by an alphabet that in- cludes *ok* and *ok* ', and by the healthiness conditions H1 and H2. For reactive programs, for instance, we have a theory of relations whose alphabets include six other observational variables, and that satisfy two other healthiness con- ditions. Alphabets and healthiness conditions are the basis to compare and combine different theories. Later on, we present a theory for angelic (and de- monic) nondeterminism; beforehand, we study set-based models for the UTP.

# Set-based models

In this section, we consider two set-based models for the UTP: relations, char- acterised by sets of pairs, and predicates transformers, with predicates charac- terised by sets. These models further clarify the role of healthiness conditions

and the internalized model of nontermination based on the observational vari- ables *ok* and *ok* '. Most importantly, however, they provide guidance in the definition of a UTP theory based on binary multirelations. It is this theory that can capture both angelic and demonic nondeterminism.

* 1. *Relations*

The set-based relational model is that of sets of pairs of states. A state associates names (of observational variables) to their values. The set *SA* of all states on an alphabet *A* contains the records with a component for each variable in *A*. Each such state is an observation of the behaviour of a program. A relation, like a UTP predicate, is a pair (*αR, R*), where *αR* is the alphabet, and *R* is a relation between the elements of *Sin*α*R* and *Sout*α*R*. Such a relation models a program by associating an observation of an initial state with each of the possible observations of a final state.

The model for abort is the universal relation: P *Sin*α × *Sout*α; when the predicate *P* (or relation *R*) is not relevant, instead of writing *inαP* (or *inαR*) and *outαP* (or *outαR*), we simply write *inα* and *outα*. Partiality models miracles. If a state is not in the domain of the relation, then it is miraculous at that state: it can achieve any required result, including *false*. In particular, the model of miracle is the empty relation.

It is not difficult to see that the first general predicate-based theory of the UTP is isomorphic to this set-based model. A simple proof is presented in [[8](#_bookmark19)]; it is based on the functions *p*2*sb* and *sb*2*p*.

Definition 3.1

' '

*p*2*sb.*(*αP, P* ) =^ (*αP,* { *s* : *Sin*α*P* ; *s* : *Sout*α*P* | *P* [*s, s /inαP, outαP* ] })

*sb*2*p.*(*αR, R*) =

^

(*αR,* E *s* : *Sin*α*R, s*' : *Sout*α*R* • (*s, s*') ∈ *R* Λ

( *x* : *inαR* • *x* = *s.x* ) Λ ( *x* : *outαR* • *x* = *s*'*.x* ))

The first, *p*2*sb*, transforms a UTP relation into a set-based relation; the sec- ond, *sb*2*p* is its inverse: it transforms a set-based relation into a UTP relation. Both *p*2*sb* and *sb*2*p* do not change the alphabet of the relations. A similar set model is used by Hoare and He when they discuss denotational semantics.

The set-based relation defined by *p*2*sb* for a predicative relation *P* is formed by pairs of states *s* and *s*' such that *P* holds when the observational variables take the values associated to them by *s* and *s*'. The predicate *P* [*s/A*] is obtained by replacing *x* with *s.x* , for all *x* in *A*.

SBH1 ∀ *s, s*' | *s.ok* = *false* • (*s, s*') ∈ *R*

SBH2 ∀ *s, s*' | (*s, s*') ∈ *R* Λ *s*'*.ok* ' = *false* • (*s, s*' ⊕ {*ok* ' '→ *true*}) ∈ *R*

SBH3 ∀ *s* | (E *s*' • *s*'*.ok* ' = *false* Λ (*s, s*') ∈ *R*) • ∀ *s*' • (*s, s*') ∈ *R*

Table 2

Set-based healthiness conditions

The predicate defined by *sb*2*p* for a relation *R* is an existential quantifica- tion over pairs of states *s* and *s*' in *R*. For each pair, a conjunction of equalities requires that each observational variable takes the value in the corresponding initial or final state. Since alphabets are finite, the conjunction is finite.

Standard work on relational semantics [[13](#_bookmark24)] singles out a special state to indicate non-termination. If an initial state is associated with all possible final states, then we cannot say whether the final state is simply arbitrary or we have a possibility of non-termination. In standard relational semantics, the model for abort that we presented above is actually the model for a program that always terminates, but whose final state is arbitrary.

The isomorphism confirms that the general UTP model is not able to capture non-termination. Hoare and He pointed out a paradox in the fact that, if the alphabet is { *x , x* ' }, then (*µ X* • *X* ); *x* := 3 is equivalent to *x* := 3. This is not really a paradox: the bottom of the lattice ⊥ is not an aborting program, but the program that terminates and gives an arbitrary value to *x* . If, in sequence, we assign 3 to *x* , then the arbitrariness is irrelevant. Their model is sensible, for terminating programs. (Their attempt to solve the supposed paradox by giving a strongest fixed point semantics to recursion was always doomed to fail. They just wanted to discuss partial correctness.) For designs, the alphabet includes *ok* and *ok* '; therefore, these variables are also part of the alphabet of the corresponding set-based relations. In Table [2](#_bookmark4),

we present healthiness conditions; we omit the obvious types of *s* and *s*'.

The healthiness condition SBH1 requires that all states *s* for which *s.ok* is false are in the domain of *R*, and are related to all possible final states. This means that a state in which the program has not started is not miraculous and leads to no controlled behaviour.

In relations that are SBH2-healthy, if a state *s* is related to a state *s*' for which *s*'*.ok* ' is false, then *s* is also related to *s*' ⊕ {*ok* ' '→ *true*}. This is the same state as *s*', except that the value of *ok* ' is *true*. This means that if it is possible not to terminate from *s*, it is also possible to terminate. Its behaviour, however, may not be completely arbitrary: it is not required that *R* relates *s* to all possible final states; this is what is required by SBH3.

The theorem below, proved in [[8](#_bookmark19)], establishes that H1, H2, and H3 corre- spond to SBH1, SBH2, and SBH3.

Theorem 3.2 *For every UTP relation* (*αP, P* ) *that is H1, p*2*sb.*(*αP, P* ) *sat- isﬁes SBH1. Conversely, for every set-based relation* (*αR, R*) *that is SBH1, sb*2*p.*(*αR, R*) *is H1. The same holds for H2 and SBH2, and for H3 and SBH3.*

We believe that it is not difficult to observe that SBH3 relations are necessarily SBH2. If the initial state *s* is related to all possible final states, then it is also related to *s*' ⊕ {*ok* ' '→ *true*}. This rather obvious result seems to be not so clear in the predicate setting. It means that, at least for the purpose of the study of total correctness of sequential programs, Hoare and He did not need to consider four healthiness conditions, but only three of them: H1, H3, and H4. It turns out, however, that non-H3 designs are important for the modelling of more sophisticated programming paradigms like CSP, for instance.

The healthiness condition H4 requires feasibility. It is not relevant for us, as miracles are an important part of Morgan’s refinement calculus and ZRC.

* 1. *Predicate transformers*

In the model of predicate transformers, we regard predicates as sets of states. The model is composed of pairs (*αPT , PT* ), where *αPT* is the alphabet of the transformer, and *PT* is a total monotonic function from P *Sout*α*PT* to P *Sin*α*PT* . A program is modelled by its weakest precondition transformer [[9](#_bookmark20)].

Isomorphisms between predicate transformers and relational models have already been studied [[12](#_bookmark23)]. The isomorphism that we propose here is similar to that in [[6](#_bookmark17)]. We define functions *sb*2*pt* and *pt*2*sb*; the first transforms a set-based relation into a weakest precondition, and the second transforms a weakest precondition back into a set-based relation. For simplicity, we ignore alphabets, which, strictly speaking, should be maintained by both functions.

Definition 3.3 *sb*2*pt.R.ψ* =^ dom(*R* −D *ψ*)

*pt*2*sb.PT* = { *s* : *Sin*α*PT* ; *s*' : *Sout*α*PT* | *s* ∈ *PT .*{ *s*' }}

In the definition of *sb*2*pt*, *ψ* is a postcondition, or rather, a set of states, which is given as argument to the predicate transformer *sb*2*pt.R*. The relation *R* −D *ψ* models all executions of *R* that do not lead to a final state that satisfies *ψ*; the operator  −D  is range subtraction. In dom(*R* −D *ψ*), we have all initial states in which it is possible not to achieve *ψ*. The complement contains all initial states in which we are guaranteed to reach a final state that satisfies *ψ*: the required weakest precondition.

PTH1 *PT .ψ* ⊆ { *s* : *Sin*α*PT* | *s.ok* = *true* } provided *ψ* /= *Sout*α*PT*

PTH3 *PT .ψ* = *PT .*{ *s*' : *ψ* | *s*'*.ok* ' = *true* } provided *ψ* /= *Sout*α*PT*

Table 3

Predicate transformers healthiness conditions

The relation *pt*2*sb.PT* associates an initial state *s* to a final state *s*' if *s* is not in the weakest precondition that guarantees that *PT* does not establish *s*'. Since it is not guaranteed that *PT* will not establish *s*', then it is possible that it will. The possibility is captured in the relation.

Since the general set-based relations can only model terminating programs, we cannot expect an isomorphism between them and the whole set of predicate transformers. In fact, they are isomorphic to the set of universally conjunctive predicate transformers *PT* : those that satisfy the property below.

(1) *PT .*( { *i* • *ψi* })= { *i* • *PT .ψi* }

An important and well-known consequence of this isomorphism is that UTP relations cannot model angelic nondeterminism. Since we have an isomor- phism between UTP relations and set-based relations, and between set-based relations and universally conjunctive predicate transformers, then UTP rela- tions are isomorphic to universally conjunctive predicate transformers.

As already said, the angelic choice in which we are interested is the least upper bound of the lattice of monotonic predicate transformers. Joins in the lattice of universally conjunctive predicate transformers are not preserved in the lattice of monotonic predicate transformers [[3](#_bookmark15)]. We need a relational model isomorphic to the monotonic predicate transformers.

Next, we investigate the set of predicate transformers that correspond to UTP designs. In this case, *ok* is in the alphabet of the states in a precon- dition, and *ok* ' is in the alphabet of the states in a postcondition. Table [3](#_bookmark5) gives healthiness conditions over such predicate transformers *PT* . The first healthiness condition, PTH1 requires that the weakest precondition for *PT* to establish any *ψ* is included in the set of initial states *s* for which *s.ok* is true. In other words, in order to guarantee a postcondition, *PT* must start. The only exception is the postcondition *Sout*α*PT* , which imposes no restrictions.

The healthiness condition PTH3 states that, in calculating *PT .ψ*, we can ignore all the states *s*' in *ψ* for which *s*'*.ok* ' is false. In other words, even if we have *s*' and *s*' ⊕ {*ok* ' '→ *true*} in *ψ*, so that termination is not required, if *PT* can guarantee *s*' or *s*' ⊕{*ok* ' '→ *true*}, then it can guarantee *s*' ⊕{*ok* ' '→ *true*}. Consequently, predicate transformers do not capture the possibility of non- termination. Again, the postcondition *Sout*α*PT* is an exception.

As expected, PTH1 and PTH3 correspond to H1 and H3 [[8](#_bookmark19)]. They restrict the behaviour of the predicate transformers for postconditions different from *Sout*α*PT* . This particular postcondition, however, is of special interest.

Universally conjunctive predicate transformers can only model terminating programs; this is because, if ([1](#_bookmark6)) holds for the empty set, then *PT .Sout*α = *Sin*α. In words, for the postcondition that does not impose any restrictions, any initial state should be satisfactory. Nevertheless, the postcondition that does not impose any restriction still requires termination. Therefore, universal conjunctivity requires that the program always terminates.

In the context of predicate transformers that involve states on *ok* and *ok* ', however, the postcondition *Sout*α does not require termination: it accepts any final state *s*', even those for which *s*'*.ok* ' = *false*. Similarly, the precondition *Sin*α does not even require the program to start.

Therefore, the universal conjunctivity of the predicate transformers corre- sponding to designs does not imply that only terminating programs can be modelled. Unfortunately, conjunctivity is still an issue: the predicate trans- formers that are PTH1 and PTH3 healthy are conjunctive. As a consequence, they cannot model angelic nondeterminism either. What we need is a rela- tional model isomorphic to monotonic, not necessarily conjunctive, predicate transformers. This is the subject of the next section.

When we consider H3-healthy designs, we get a model isomorphic to stan- dard weakest preconditions; in [[8](#_bookmark19)] we present an isomorphism between the predicate transformers above and those on postconditions and preconditions that do not refer to *ok* and *ok* '. In [[10](#_bookmark21)], different healthiness conditions that lead to a theory of general correctness are proposed.

# Binary Multirelations

A relational model isomorphic to monotonic predicate transformers is pre- sented in [[20](#_bookmark30)]; in that work, the relations are called binary multirelations. In our setting, we define a binary multirelation as a pair (*αBM , BM* ), where *αBM* is an alphabet, and *BM* is a relation between *Sin*α*BM* and postcondi- tions: elements of P *Sout*α*BM* . Intuitively, *BM* captures the behaviour of a program by associating each initial state with all the postconditions that the program can angelically choose to satisfy.

If a postcondition *ψ* can be satisfied, so can all postconditions weaker than

*ψ*. Therefore, we have the following healthiness condition.

BMH ∀ *s, ψ*1*, ψ*2 | (*s, ψ*1) ∈ *BM* Λ *ψ*1 ⊆ *ψ*2 • (*s, ψ*2) ∈ *BM*

The model for abort, for example, is the empty relation. Miracle relates each

initial state with every subset of *Sout*α; it is the universal relation. The binary multirelation for an assignment *x* := *e* relates every initial state *s* with every set that includes [*s*]'⊕{*x* ' '→ *e*}. This is a final state in which the value for each variable *v* ' of *outα* is *s.v* , except for *x* ', whose value is *e*. If executed in *s*, then *x* := *e* reaches a final state that satisfies the postcondition { [*s*]' ⊕ {*x* ' '→ *e*} }, and any other weaker postcondition represented by one of its supersets.

The binary multirelation that models the angelic choice *x* := 0 H *x* := 1 is { *s,ψ* | {(*x* ' '→ 0)} ⊆ *ψ* V {(*x* ' '→ 1)} ⊆ *ψ* }. It associates to each initial state *s* the postconditions that include (*x* ' '→ 0) or (*x* ' '→ 1). We use (*x* ' '→ *v* ) to denote a record with a single component named *x* ' whose value is *v* . This is because the angel can ensure the final value of *x* to be either 0 or 1, as required. For the demonic choice, *x* := 0 H *x* := 1, the range of the binary multirelation includes the supersets of { (*x* ' '→ 0)*,* (*x* ' '→ 1) }. In this case, the demon is in control: the final value of *x* is arbitrarily chosen to be 0 or 1.

For *x* := 0 H (*x* := 1 H *x* := 2), which involves a demonic and an angelic choice, the model is { *s,ψ* | {(*x* ' '→ 0)*,* (*x* ' '→ 1)} ⊆ *ψ* V {(*x* ' '→ 0)*,* (*x* ' '→ 2)} ⊆ *ψ* }. If either 0 or 1 is an acceptable final value for *x* , then the angel can help. Similarly, if 0 and 2 are acceptable, we are guaranteed success. Nevertheless, 1 or 2 only cannot be guaranteed; of course, a requirement for 0, 1, or 2 is successful, and indeed the postcondition { (*x* ' '→ 0)*,* (*x* ' '→ 1)*,* (*x* ' '→ 2) } is a superset of both { (*x* ' '→ 0)*,* (*x* ' '→ 1) } and { (*x* ' '→ 0)*,* (*x* ' '→ 2) } and therefore is included in the range of the binary multirelation.

Here, we consider the isomorphism between binary multirelations and pred- icate transformers characterised by the functions below.

Definition 4.1 *bm*2*pt.BM .ψ* = { *s* | (*s, ψ*) ∈ *BM* }

*pt*2*bm.PT* = { (*s, ψ*) | *s* ∈ *PT .ψ* }

The function *bm*2*pt* converts a binary multirelation to a weakest precondi- tion: *bm*2*pt.BM* is guaranteed to establish *ψ* in all initial states *s* associated to *ψ* in *BM* ; in these states *BM* angelically chooses to establish *ψ* if required. Conversely, the multirelation *pt*2*bm.PT* associates an initial state *s* with all the postconditions that *PT* is guaranteed to establish from *s*.

This isomorphism is simpler than that in [[20](#_bookmark30)], which constructs the bi- nary multirelation corresponding to a predicate transformer using prime filter representations of states. Our proof that the functions *bm*2*pt* and *pt*2*bm* characterise an isomorphism is very simple.

Theorem 4.2 *pt*2*bm.*(*bm*2*pt.BM* )= *BM*

Proof.

*pt*2*bm.*(*bm*2*pt.BM* ) [definition of *pt*2*bm*]

= { (*s, ψ*) | *s* ∈ *bm*2*pt.BM .ψ* } [definition of *bm*2*pt*]

= { (*s, ψ*) | *s* ∈ { *s* | (*s, ψ*) ∈ *BM* }} [property of set comprehension]

= { (*s, ψ*) | (*s, ψ*) ∈ *BM* } [property of sets]

= *BM*

Theorem 4.3 *bm*2*pt.*(*pt*2*bm.PT* )= *PT*

Proof.

*bm*2*pt.*(*pt*2*bm.BM* )*.ψ* [definition of *bm*2*pt*]

= { *s* | (*s, ψ*) ∈ *pt*2*bm.PT* } [definition of *pt*2*bm*]

= { *s* | (*s, ψ*) ∈ { (*s, ψ*) | *s* ∈ *PT .ψ* }} [property of set comprehension]

= { *s* | *s* ∈ *PT .ψ* } [property of sets]

= *PT .ψ*

The following two theorems establish that monotonic predicate transformers correspond to BMH-healthy multirelations. They conclude our argument; we have a model isomorphic to monotonic predicate transformers.

Theorem 4.4 *For a BMH-healthy binary relation BM , bm*2*pt.BM is mono- tonic.*

Proof. Let *ψ*1 and *ψ*2 be such that *ψ*1 ⊆ *ψ*2.

*bm*2*pt.BM .ψ*1 [definition of *bm*2*pt*]

= { *s* | (*s, ψ*1) ∈ *BM* } [*BM* is healthy and *ψ*1 ⊆ *ψ*2]

⊆ { *s* | (*s, ψ*2) ∈ *BM* } [definition of *bm*2*pt*]

= *bm*2*pt.BM .ψ*2

Theorem 4.5 *For a monotonic PT , the binary multirelation pt* 2*bm.PT is*

*BMH-healthy.*

Proof. Let *ψ*1 and *ψ*2 be such that *ψ*1 ⊆ *ψ*2.

(*s, ψ*1) ∈ *pt*2*bm.PT* [definition of *pt*2*bm*]

***pt2bm sb2pt p2sb***

binary

multirelations

predicate

transformers

Set-based

relations

UTP

predicates

***bm2pt***

***pt2sb***

***sb2p***

Fig. 1. Models and isomorphisms

= (*s, ψ*1) ∈ { (*s, ψ*) | *s* ∈ *PT .ψ* } [property of set comprehension]

= *s* ∈ *PT .ψ*1 [*PT* is monotonic and *ψ*1 ⊆ *ψ*2]

⇒ *s* ∈ *PT .ψ*2 [property of set comprehension]

= (*s, ψ*2) ∈ { (*s, ψ*) | *s* ∈ *PT .ψ* } [definition of *pt*2*bm*]

= (*s, ψ*2) ∈ *pt*2*bm.PT*

What we need now is a way of expressing binary multirelations as alphabetised predicates in the style of the UTP.

* 1. *Predicative theory*

The key point to define a UTP theory based on binary multirelations is the choice of alphabet. We propose a view of a binary multirelations as a relation between a state on an alphabet *inα* and a state on { *dc*' }. The value of *dc*' is the set of demonic choices available to the program: a set of states on an alphabet *outα*, or yet, a postcondition. For example, in a theory of designs in which we can handle angelic nondeterminism, the alphabet is { *v, ok, dc*' }, where *v* stands for the list of program variables. In *dc*', the states are records that give values to the variables *v* ' and *ok* '.

Figure [1](#_bookmark8) summarises the isomorphisms we have defined so far. We are looking for a way of representing binary multirelations as UTP predicates. We cannot use *pt*2*sb* because it cannot handle non-conjunctive predicate trans- formers. Instead, we define an isomorphism between binary multirelations and set-based relations with alphabet *inα* ∪{ *dc*' }.

Definition 4.6

*bm*2*sb.BM* = { *s* : *Sin*α; *s*' : *S*{*dc*'} | (*s, s*'*.dc*') ∈ *BM* }

*sb*2*bm.DCR* = { *s* : *Sin*α; *ss* : P *Sout*α | (*s,* (*dc*' '→ *ss*)) ∈ *DCR* }

Using *bm*2*sb*, we get a standard set-based relation in which the sets in the range of the original binary multirelation are wrapped in records with a single

***sb2bm***

***p2sb***

binary multirelations

predicate transformers

***pt2r***

Set-based relations

UTP

predicates

***sb2p***

***bm2pt pt2sb***

***pt2bm sb2pt***

***bm2sb***

Fig. 2. Extra isomorphism

component *dc*'; the function *sb*2*bm* unwraps these records. The proof that

*bm*2*sb* and *sb*2*sm* establish an isomorphism is trivial.

We actually aim at expressing predicate transformers as UTP predicates using *pt*2*bm*, *bm*2*sb*, and *sb*2*p*. In our calculations, we name the composition

of *pt*2*bm* and *bm*2*sb* as *pt*2*r* =^ *bm*2*sb* ◦ *pt*2*bm*. The next theorem gives a

direct characterisation of *pt*2*r* .

Theorem 4.7 *pt*2*r.PT* = { *s* : *Sin*α; *s*' : *S*{*dc*'} | *s* ∈ *PT .*(*s*'*.dc*') }

For conciseness, we omit its simple proof. Figure [2](#_bookmark9) shows the additional isomorphism and function that we use in the sequel.

For example, the transformer *abort* maps all postconditions to the empty set: it can never guarantee anything. In the UTP, it corresponds to false.

Theorem 4.8 *sb*2*p.*(*pt*2*r.abort*)= false*.*

Proof.

*sb*2*p.*(*pt*2*r.abort*) [definition of *pt*2*r* ]

= *sb*2*p.*{ *s, s*' | *s* ∈ *abort.*(*s*'*.dc*') } [definition of *abort*]

= *sb*2*p.*∅ [definition of *sb*2*p*]

= E *s, s*' • (*s, s*') ∈ ∅ Λ ( *x* : *inα* • *x* = *s.x* ) Λ *dc*' = *s*'*.dc*'

[property of sets]

= *false*

Therefore, partiality models abortion. The miraculous program is true.

* 1. *Healthiness condition*

In the UTP, the healthiness condition for binary multirelations is as follows.

PBMH *P* ; *dc* ⊆ *dc*' = *P*

This requires that, if, after executing *P* , we execute a program that enlarges *dc*', then the result could have been obtained by *P* itself. A healthy *P* char- acterises *dc*' not by defining a particular value for it, but the smallest set of elements it should include. All the supersets should be allowed.

Healthy binary multirelations correspond to PBMH-healthy predicates.

Theorem 4.9 *If BM satisﬁes BMH, then sb*2*p.*(*bm*2*sb.BM* ) *satisﬁes PBMH.*

Proof.

*sb*2*p.*(*bm*2*sb.BM* ); *dc* ⊆ *dc*' [definition of *bm*2*sb*]

= *sb*2*p.*{ *s* : *Sin*α; *s*' : *S*{*dc*'} | (*s, s*'*.dc*') ∈ *BM* }; *dc* ⊆ *dc*'

[definition of *sb*2*p*]

= (E *s, s*' •

[property of set comprehension]

(*s, s*') ∈ { *s* : *Sin*α; *s*' : *S*{*dc*'} | (*s, s*'*.dc*') ∈ *BM* }Λ

( *x* : *inαR* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*');

*dc* ⊆ *dc*'

= (E *s, s*' • (*s, s*'*.dc*') ∈ *BM* Λ ( *x* : *inα* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*');

*dc* ⊆ *dc*'

[definition of sequential composition]

= E *s, s*'*, dc*0 •

[predicate calculus]

(*s, s*'*.dc*') ∈ *BM* Λ ( *x* : *inα* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*0 Λ *dc*0 ⊆ *dc*'

= E *s, s*' • (*s, s*'*.dc*') ∈ *BM* Λ ( *x* : *inα* • *x* = *s.x* ) Λ *s*'*.dc*' ⊆ *dc*'

[*BM* is BMH-healthy and predicate calculus]

= E *s, s*' • (*s, dc*') ∈ *BM* Λ ( *x* : *inα* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*'

[predicate calculus]

= E *s, s*' • (*s, s*'*.dc*') ∈ *BM* Λ ( *x* : *inα* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*'

[property of sets]

= E *s, s*' • [definitions of *sb*2*p* and *bm*2*sb*]

(*s, s*') ∈ { *s* : *Sin*α; *s*' : *S*{*dc*'} | (*s, s*'*.dc*') ∈ *BM* }Λ

( *x* : *inα* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*'

= *sb*2*p.*(*bm*2*sb.BM* )

Theorem 4.10 *If P is a PBMH-healthy predicate, then sb*2*bm.*(*p*2*sb.P* ) *is*

*BMH-healthy.*

Proof. Let *ψ*1 and *ψ*2 be such that *ψ*1 ⊆ *ψ*2.

(*s, ψ*1) ∈ *sb*2*bm.*(*p*2*sb.P* ) [definition of *p*2*sb*]

= (*s, ψ*1) ∈ *sb*2*bm.*{ *s, s*' | *P* [*s, s*'*/inα, dc*'] } [definition of *sb*2*bm*]

= (*s, ψ*1) ∈ [property of sets]

{ *s* : *Sin*α; *ss* : P *S*{*dc*'} | (*s,* (*dc*' '→ *ss*)) ∈ { *s, s*' | *P* [*s, s*'*/inα, dc*'] }}

= (*s,* (*dc*' '→ *ψ*1)) ∈ { *s, s*' | *P* [*s, s*'*/inα, dc*'] } [property of sets]

= *P* [*s, ψ*1*/inα, dc*'] [*P* is PBMH-healthy]

= (*P* ; *dc* ⊆ *dc*')[*s, ψ*1*/inα, dc*'] [substitution]

= *P* [*s/inα*]; *dc* ⊆ *ψ*1 [definition of sequential composition]

= E *dc*0 • *P* [*s, dc*0*/inα, dc*'] Λ *dc*0 ⊆ *ψ*1 [*ψ*1 ⊆ *ψ*2]

⇒ E *dc*0 • *P* [*s, dc*0*/inα, dc*'] Λ *dc*0 ⊆ *ψ*2

[definition of sequential composition, and substitution]

= (*P* ; *dc* ⊆ *dc*')[*s, ψ*2*/inα, dc*'] [*P* is PBMH-healthy]

= *P* [*s, ψ*2*/inα, dc*'] [definitions of *p*2*sb* and *sb*2*bm*]

= (*s, ψ*2) ∈ *sb*2*bm.*(*p*2*sb.P* )

It is pleasing that the healthiness condition can be cast in a quite simple way, and also in terms of the fixpoint of the idempotent PHBM(*X* )= *X* ; *dc* ⊆ *dc*'. This is important for the approach to linking theories encouraged by the UTP.

* 1. *Reﬁnement*

The refinement relation is implication; not reverse implication as in the UTP. Still, it is just implication, and, more importantly, it corresponds to refinement in the predicate transformer model.

As usual, we define predicate transformer refinement as follows.

Definition 4.11 *PT*1 ±*PT PT*2 =^ ∀ *ψ* • *PT*1*.ψ* ⊂ *PT*2*.ψ*

For healthy binary multirelations, we have the following definition.

Definition 4.12 *BM*1 ±*BM BM*2 =^ *BM*1 ⊆ *BM*2

The next theorem establishes that these orders are compatible.

Theorem 4.13 *BM*1 ±*BM BM*2 *if, and only if, bm*2*pt.BM*1 ±*PT bm*2*pt.BM*2*.*

Proof.

*bm*2*pt.BM*1 ±*PT bm*2*pt.BM*2 [definition of ±*PT* ]

= ∀ *ψ* • *bm*2*pt.BM*1*.ψ* ⊆ *bm*2*pt.BM*2*.ψ* [definition of *bm*2*pt*]

= ∀ *ψ* • { *s* | (*s, ψ*) ∈ *BM*1} ⊆ { *s* | (*s, ψ*) ∈ *BM*2} [property of sets]

= ∀ *ψ, s* • (*s, ψ*) ∈ *BM*1 ⇒ (*s, ψ*) ∈ *BM*2 [property of sets]

= *BM*1 ±*BM BM*2

Finally, we define angelic refinement in the UTP theory.

Definition 4.14 *P* ±*A Q* =^ [ *P* ⇒ *Q* ]

The correspondence between this refinement relation and that for binary mul- tirelations is established below.

Theorem 4.15 *P* ±*A Q if, and only if, sb*2*bm.*(*p*2*sb.P* ) ±*BM sb*2*bm*(*p*2*sb.Q* )*.*

Proof.

*sb*2*bm.*(*p*2*sb.P* ) ±*BM sb*2*bm*(*p*2*sb.Q* ) [definition of ±*BM* ]

= *sb*2*bm.*(*p*2*sb.P* ) ⊂ *sb*2*bm*(*p*2*sb.Q* ) [property of sets]

= ∀ *s,ψ* • (*s, ψ*) ∈ *sb*2*bm.*(*p*2*sb.P* ) ⇒ (*s, ψ*) ∈ *sb*2*bm.*(*p*2*sb.Q* )

[definition of *sb*2*bm*]

= ∀ *s,ψ* • (*s, dc*' '→ *ψ*) ∈ *p*2*sb.P* ⇒ (*s, dc*' '→ *ψ*) ∈ *p*2*sb.Q*

[definition of *p*2*sb*]

= 6 *s,ψ* • *P* [*s, ψ/inα, dc*'] ⇒ *Q* [*s, ψ/inα, dc*'] [predicate calculus]

= 6 *x* : *inα, dc*' • *P* ⇒ *Q* [the alphabet is *inα* ∪ {*dc*'}]

= [*P* ⇒ *Q* ]

The pre-order proposed in [[20](#_bookmark30)] for binary multirelations becomes a partial or- der in the restricted setting of healthy binary multirelations. Also, it collapses to set inclusion, which is the order we adopt here.

* 1. *Operators*

Angelic choice *P* H *Q* is characterised by disjunction. The program *P* H *Q* gives all the guarantees that can be provided by choosing *P* , together with those that arise from the possibility of choosing *Q* .

Theorem 4.16 *sb*2*p.*(*pt*2*r.*(*P* H *Q* )) = *sb*2*p.*(*pt*2*r.P* ) V *sb*2*p.*(*prt*2*.Q* )

Proof.

*sb*2*p.*(*pt*2*r.*(*P* H *Q* )) [Theorem [4.7](#_bookmark10)]

= *sb*2*p.*{ *s, s*' | *s* ∈ (*P* H *Q* )*.*(*s*'*.dc*'[)p}redicate transformer semantics of H]

= *sb*2*p.*{ *s, s*' | *s* ∈ *P.*(*s*'*.dc*') ∪ *Q.*(*s*'*.dc*') } [definition of *sb*2*p*]

= E *s, s*' •

[property of sets and predicate calculus]

*s* ∈ *P.*(*s*'*.dc*') ∪ *Q.*(*s*'*.dc*') Λ ( *x* : *inαR* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*'

= (E *s, s*' • *s* ∈ *P.*(*s*'*.dc*') Λ ( *x* : *inαR* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*') V

(E *s, s*' • *s* ∈ *Q.*(*s*'*.dc*') Λ ( *x* : *inαR* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*')

[definitions of *pt*2*r* and *sb*2*p*]

= *sb*2*p.*(*pt*2*r.P* ) V *sb*2*p.*(*pt*2*r.Q* )

Demonic choice is captured by conjunction; a postcondition is guaranteed by

*P* H *Q* only if both *P* and *Q* can guarantee it.

Theorem 4.17 *sb*2*p.*(*pt*2*r.*(*P* H *Q* )) = *sb*2*p.*(*pt*2*r.P* ) Λ *sb*2*p.*(*prt*2*.Q* )

Proof.

*sb*2*p.*(*pt*2*r.*(*P* H *Q* )) [Theorem [4.7](#_bookmark10)]

= *sb*2*p.*{ *s, s*' | *s* ∈ (*P* H *Q* )*.*(*s*'*.dc*'[)p}redicate transformer semantics of H]

= *sb*2*p.*{ *s, s*' | *s* ∈ *P.*(*s*'*.dc*') ∩ *Q.*(*s*'*.dc*') } [definition of *sb*2*p*]

= E *s, s*' •

[property of sets and predicate calculus]

*s* ∈ *P.*(*s*'*.dc*') ∩ *Q.*(*s*'*.dc*') Λ ( *x* : *inαR* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*'

= (*θx* : *inα* • *x* '→ *x* ) ∈ *P.dc*' ∩ *Q.dc*' [property of sets]

= (*θx* : *inα* • *x* '→ *x* ) ∈ *P.dc*' Λ (*θx* : *inα* • *x* '→ *x* ) ∈ *Q.dc*'

[property of sets and predicate calculus]

(E *s, s*' • *s* ∈ *P.*(*s*'*.dc*') Λ ( *x* : *inαR* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*') Λ

(E *s, s*' • *s* ∈ *Q.*(*s*'*.dc*') Λ ( *x* : *inαR* • *x* = *s.x* ) Λ *s*'*.dc*' = *dc*')

[definitions of *pt*2*r* and *sb*2*p*]

= *sb*2*p.*(*pt*2*r.P* ) Λ *sb*2*p.*(*pt*2*r.Q* )

In this proof, the notation (*θx* : *A* • *x* '→ *v* ) describes the record that asso- ciates each name *x* in the alphabet *A* to a value *v* . Above, the value is that of the variable *x* itself; we have a predicate on the variables *x* and *dc*'.

Sequential composition cannot correspond to relational composition. It uses the operator ∗ to lift *Q* to a predicate on *dc* and *dc*'. It is inspired on the UTP treatment of logic programming, and is defined as follows.

*Q* =^

∗

*µ X* • true D *dc* = ∅ D var *s* • *s*' ∈ *dc*;

(*v* := *s.v* ; *Q* ) H (*dc* := *dc* \{ *s* }; *X* )

end

The behaviour of *P* followed by *Q* in sequence is given by *P* ; *Q* ∗: after the execution of *P* , *Q* ∗ recursively selects a state in *dc*' and executes *Q* . The program *P* D *c* D *Q* is a conditional: it executes *P* if *c* holds, else it executes

*Q* . A variable *s* is declared to hold a state in *dc*. The observational variables are initialised as in *s* before *Q* is executed. The demonic choice of all the outcomes of the executions of *Q* is the result of the sequence.

It is unavoidable that the definitions of some operators are more compli- cated than those in the original UTP model. It is part of the philosophy of the UTP to study constructs and concepts in isolation: we have provided a theory for angelic nondeterminism which can be incorporated to the other theories as needed. We have also established that we do need a more elaborate relational model to capture angelic nondeterminism.

# Conclusions

The central objective of Hoare and He’s unifying theories of programming is to formalise different programming paradigms within a common semantic framework, so that they may be directly compared and new compound pro- gramming languages and refinement calculi may be developed. This ambitious research programme has only just been started. An important question to ask is: what are the theoretical limits to this investigation?

Angelic nondeterminism is a valuable concept: it plays an important roˆle in refinement calculi, and it is used as an abstraction in search-based and constraint-oriented programming, hiding details of how particular strategies are implemented. Our main contribution is a predicative account of binary multirelations that unifies angelic nondeterminism into the UTP.

We describe the predicative theory of alphabetised relations and the the- ory of designs, where we can observe the start and termination of a program. Designs enable reasoning about total correctness, and a set-based model of relations brings this fact sharply into focus. We show that there is an isomor- phism between our set-based relations and universally conjunctive predicate transformers. This establishes a connection with an existing result: conjunc- tive predicate transformers cannot capture angelic nondeterminism.

A relational model that can capture both angelic and demonic nondeter- minism is presented in [[20](#_bookmark30)]. We cast that model in the UTP predicative style, including a healthiness condition and the refinement relation. This allows its use in an integrated framework that covers, for instance, concurrency and higher-order programming. We are going to use this model to extend the existing semantics of our combined formalism [[21](#_bookmark32)], and prove refinement laws.

In [[4](#_bookmark14)], Back and von Wright present another relational model isomorphic to predicate transformers; it is a functional model called choice semantics. In that work, a program *P* is a function from initial states *s* to the set of postconditions that can be satisfied when *P* is executed in *s*. The choice semantics is, of course, isomorphic to binary multirelations. Since, in the predicative style of the UTP, relations are defined punctually, it was more convenient to base our work on binary multirelations.

The work in [[16](#_bookmark28)] presents a functional semantics for a tactic language which includes angelic nondeterminism. The semantics of angelic choice is a list that contains all the options available to the angel; demonic nondeterminism is not included. In [[17](#_bookmark31)], the set-based model of binary relations is used to support angelic and demonic nondeterminism in a calculus for functional programs. They adopt two refinement relations, one of which is the same as ours.

Both [[20](#_bookmark30)] and [[17](#_bookmark31)] present operations that model, for example, angelic non- determinism and sequence as operations on sets. Our contribution is to cast these operations at the level of UTP predicates, where they can be integrated into more powerful theories of programming.

# References

1. Back, R. J. R. and J. Wright, *A Lattice-theoretical Basis for a Speciﬁcation Language*, in:

J. L. A. van de Snepscheut, editor, *Mathematics of Program Construction: 375th Anniversary of the Groningen University*, LNCS 375 (1989), pp. 139 – 156.

1. Back, R. J. R. and J. Wright, *Duality in Speciﬁcation Languages: A Lattice-theoretical Approach*, Acta Informatica 27 (1990), pp. 583 – 625.
2. Back, R. J. R. and J. Wright, *Combining angels, demons and miracles in program speciﬁcations*, Theoretical Computer Science 100 (1992), pp. 365 – 383.
3. Back, R. J. R. and J. Wright, “Refinement Calculus: A Systematic Introduction,” Graduate Texts in Computer Science, Springer-Verlag, 1998.
4. Cavalcanti, A. L. C., A. C. A. Sampaio and J. C. P. Woodcock, *A Reﬁnement Strategy for*

*Circus*, Formal Aspects of Computing 15 (2003), pp. 146 — 181.

1. Cavalcanti, A. L. C. and J. C. P. Woodcock, *A Weakest Precondition Semantics for Z*, The Computer Journal 41 (1998), pp. 1 – 15.
2. Cavalcanti, A. L. C. and J. C. P. Woodcock, *ZRC—A Reﬁnement Calculus for Z*, Formal Aspects of Computing 10 (1999), pp. 267—289.
3. Cavalcanti, A. L. C. and J. C. P. Woodcock, *Angelic Nondeterminism and Unifying Theories of Programming (Extended Version)*, Technical report, University of Kent - Computing Laboratory (2004).
4. Dijkstra, E. W., “A Discipline of Programming,” Prentice-Hall, 1976.
5. Dunne, S., *Recasting Hoare and He’s Unifying Theories of Programs in the Context of General* *Correctness*, in: A. Butterfield and C. Pahl, editors, *IWFM’01: 5th Irish Workshop in Formal Methods*, BCS Electronic Workshops in Computing, Dublin, Ireland, 2001.
6. Gardiner, P. H. B. and C. C. Morgan, *Data Reﬁnement of Predicate Transformers*, Theoretical Computer Science 87 (1991), pp. 143 – 162.
7. Hesselink, W. H., “Programs, Recursion and Unbounded Choice – Predicate Transformation Semantics and Transformation Rules,” Cambridge Tracts in Theoretical Computer Science 27, Cambridge University Press, 1992.
8. Hoare, C. A. R. and J. He, *The Weakest Prespeciﬁcation*, Technical Monograph TM-PRG-44, Oxford University Computing Laboratory, Oxford – UK (1985).
9. Hoare, C. A. R. and H. Jifeng, “Unifying Theories of Programming,” Prentice-Hall, 1998.
10. Jagadeesan, R., V. Shanbhogue and V. Saraswat, *Angelic non-determinism in concurrent constraint programming*, Technical report, Xerox Park (1991).
11. Martin, A. P., P. H. B. Gardiner and J. C. P. Woodcock, *A Tactical Calculus*, Formal Aspects of Computing 8 (1996), pp. 479–489.
12. Martin, C. E., S. A. Curtis and I. Rewitzky, *Modelling Nondeterminism*, in: *Mathematics of Program Construction*, LNCS, 2004, pp. 228 – 251.
13. Morgan, C. C., “Programming from Specifications,” Prentice-Hall, 1994, 2nd edition.
14. Morgan, C. C. and P. H. B. Gardiner, *Data Reﬁnement by Calculation*, Acta Informatica 27

(1990), pp. 481—503.

1. Rewitzky, I., *Binary Multirelations*, in: H. Swart, E. Orlowska, G. Schmidt and M. Roubens, editors, *Theory and Application of Relational Structures as Knowledge Instruments*, LNCS 2929, 2003, pp. 256 – 271.
2. Woodcock, J. C. P. and A. L. C. Cavalcanti, *The Semantics of Circus*, in: D. Bert, J. P. Bowen,

M. C. Henson and K. Robinson, editors, *ZB 2002: Formal Speciﬁcation and Development in Z and B*, LNCS 2272 (2002), pp. 184—203.

1. Woodcock, J. C. P. and A. L. C. Cavalcanti, *A Tutorial Introduction to Designs in Unifying Theories of Programming*, in: E. A. Boiten, J. Derrick and G. Smith, editors, *IFM 2004: Integrated Formal Methods*, LNCS 2999 (2004), pp. 40 – 66, invited tutorial.