Angelic Nondeterminism and Unifying Theories of Programming (Extended Version)

Ana Cavalcanti and Jim Woodcock

Technical Report No. 13-04
June 2004

Angelic Nondeterminism and Unifying Theories of Programming (Extended Version)

Ana Cavalcanti and Jim Woodcock

Computing Laboratory
University of Kent

Abstract. Hoare and He have proposed unifying theories of programming (UTP): a model of alphabetised relations expressed as predicates, which provides support for program development in a number of programming paradigms. Their objective is the unification of languages and techniques, so that developers can benefit from results of works that were, until then, conflicting in their approach. They consider fundamental programming concepts like nondeterminism, specification as pre and postcondition pairs, and concurrency; however, they leave the study of many constructs open. In this report, we investigate the possibility of unifying angelic nondeterminism into the UTP. We consider its general relational setting, and the more restricted setting of designs, which is the basis for all the models considered by Hoare and He. We study isomorphic models based on relations between sets of states and on predicate transformers. We conclude that the introduction of angelic nondeterminism requires a more elaborate relational framework. We proceed to 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.

1 Introduction

Angelic nondeterminism is a specification and programming concept that is typically available in unified languages of refinement calculi [17, 4], and in concurrent constraint programming languages [14]. In program development techniques, it is reflected in choice constructs in which the choice is not arbitrary, but made to guarantee success, if possible. In programming languages, it is reflected in the use of backtracking in exhaustive searches. The work in [15] explores angelic nondeterminism in a language for definition of proof tactics.

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

In [10], Gardiner and Morgan identify angelic choice with the least upper bound in the lattice of monotonic predicate transformers. In [18], they use this construct to define logical constants, which are pervasive in refinement techniques, and are sometimes named logical, auxiliary, or angelic variables. The logical constants play a fundamental rôle 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 Morgan's refinement calculus [17], 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. Furthermore, in that technique, logical constants are central to the stepwise calculational development of sequences and loops.

Back and von Wright's work on refinement [4] has also explored the use of angelic nondeterminism. They have extensively studied the set of monotonic predicate transformer as a lattice with the refinement ordering. They have identified interesting sublattices, in which choice can be either angelic or demonic, and a complete base language, which can describe any monotonic predicate transformer [1, 2]. More recently, they have suggested the use of angelic nondeterminism to model user interactions with a system, and game-like situations.

Morgan's refinement calculus has been extended and adapted to handle Z specifications [24]; the resulting calculus is called ZRC [7]. It is incorporated in *Circus* [22], a combination of Z and CSP [20] that supports refinement of state-rich, reactive programs. The design of *Circus* follows the recent trend to combine notations to provide more powerful support for program design. It has been successfully applied in a number of case studies, and has a refinement theory and strategy that supports decomposition of the state and behaviour of centralised systems [21, 5].

Departing from standard work in refinement calculi, the semantics of *Circus* is Hoare and He's unifying theories of programming (UTP) [13]. This is a relational model for program development that is used to link constructs available in several programming paradigms: imperative, concurrent, logical, and others.

In the UTP, a program is modelled as an alphabetised relation, represented as a predicate over a number of observational variables. Each observational variable records information relevant to characterise the behaviour of a program. For example, program variables are observational variables, and for a reactive program, we also have an observational variable tr that records the sequence of its interactions with the environment. In the predicate model, 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. Here, we are only concerned with observation after termination.

Hoare and He introduce a general theory of relations as a model for a simple imperative language. Later, an observational variable ok is introduced; it is used to characterise a theory for a particular form of relation: designs. These correspond to pre and postcondition pairs, and are taken as the basis for the study of total correctness. Additional observational variables are added further ahead to capture constructs like communication and concurrency. At the core of the work is the notion of refinement, which is captured as implication in all theories.

By providing an integrated 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. Nevertheless, logical constants and, more generally, angelic nondeterminism are not considered. Since in *Circus* 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 in the context of weakest precondition semantics. Moreover, there are results on the relationship between relational and predicate transformer models, using a view of relations as sets of pairs of states, and of predicates as sets of states [11, 6]. Consequently, firstly we consider a set-based relational model that we prove isomorphic to the UTP model. This alternative model is rather simple, and its connection to the UTP is trivial. Surprisingly, however, it is useful in clarifying some of the restrictions of the UTP discussed by Hoare and He. We use it to explain that the theory of designs encompasses more programs than those usually considered in the theories of total correctness. Based on this observation, we suggest that one of the healthiness conditions that characterise a design is not necessary.

Secondly, we propose a predicate transformer model, which we prove isomorphic to the set-based relational model, and, consequently, to the UTP model. A study of the properties of the predicate transformer model shows that angelic nondeterminism cannot be directly represented in the theory of designs. This negative result could only be obtained using a model that can be used to reason about angelic nondeterminism; predicate transformers was an obvious choice. An alternative model is presented; it is based on the binary multirelations introduced in [19].

In the next section, we present an overview of the unifying theories of programming; we concentrate our description on the general approach to relations, and on the theory of designs. In Section 3, we consider the set-based isomorphic relational model; in Section 4 we present the predicate transformer model, and explain that angelic nondeterminism is not immediately available in the UTP. Binary multirelations are considered in Section 5. Finally, in Section 6 we present our conclusions and directions for future work.

2 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; in 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 regarded as a pair $(\alpha P, P)$, 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 $in\alpha P$ of undashed names of the observational variables, and the set $out\alpha P$ of dashed names. The free variables of P must be contained in αP.

In an homogenous relation, $out\alpha P = in\alpha' P$, where $in\alpha' P$ is the set obtained by dashing all the variables in $in\alpha P$. This means that the observations made before the program starts are the same made at a later stage.

As an example, we present the model of an assignment $x := e$, assuming that the observational variables are x, y, and z.

$$x := e \;\widehat{=}\; (x' = e \wedge y' = y \wedge z' = z)$$

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 $I\!I$ skips: it does not change the observational variables.

$$I\!I \;\widehat{=}\; (v' = v)$$

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.

$$P(v') \,;\, Q(v) \;\widehat{=}\; \exists\, v_0 \bullet P(v_0) \wedge Q(v_0) \quad \text{provided } out\alpha P = in\alpha' Q = \{v'\}$$

The notation $P(v')$ emphasises that P may have free occurrences of observational variables v'; the later reference to $P(v_0)$ refers to the predicate obtained by substituting v_0 for the free occurrences of v' in P. Similarly, for $Q(v)$ and $Q(v_0)$.

Nondeterministic choice is demonic.

$$P \sqcap Q \;\widehat{=}\; P \vee Q$$

It behaves like either P or Q, independently of the possibilities of success.

The set of relations with a particular alphabet is a complete lattice, with order $\Leftarrow$; this is the refinement ordering in this setting. More formally, the program denoted by P is refined by that denoted by Q when $[Q \Rightarrow P]$. 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. For simplicity, we are ignoring alphabets, which must be the same for P and Q. The square brackets denote universal quantification over all the alphabet.

The least upper bound $\sqcap S$ of a set S of relations is defined algebraically.

$$[P \Leftarrow \sqcap S] \;\widehat{=}\; ([P \Leftarrow X] \text{ for all } X \text{ in } S)$$

The bottom of this lattice is the program $\perp$, which is called abort.

$$\perp \;\widehat{=}\; \textbf{true}$$

Incidentally, the top element is **false**; it is written $\top$ 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 $\mu\; X \bullet F(X)$. This is the least fixed point of the function F.

Hoare and He point out what they regard as an infelicity of this model. The recursive program $\mu X \bullet X$ is supposed to model an infinite loop; it is actually equivalent to $\bot$ or **true**. Nonetheless, the sequence $(\mu X \bullet X) \; ; x' = 3$ is equivalent to $x' = 3$, even though it should not really be possible to recover from a program that does not terminate.

As a solution to this supposed paradox, Hoare and He investigate the possibility of modelling recursion using strongest fixed points. This, however, does not solve the problem. In this case $\mu X \bullet X$ is **false**, but $\mathbf{false} \sqcap P = P$. In other words, if $\mu X \bullet X$ is a non-terminating loop, choice entails the exclusion of any non-terminating options; it is angelic in the sense that it guarantees termination whenever possible. This, however, is not the intended meaning.

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 $P \vdash Q$.

$$(P \vdash Q) \;\widehat{=}\; (ok \wedge P) \Rightarrow (ok' \wedge Q)$$

The predicates P and Q are not supposed to refer to ok and ok'; they 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' stand for the observational variables other than x and x'.

$$\begin{array}{l} x := e \;\widehat{=}\; \mathbf{true} \vdash x' = e \wedge y' = y \\ I\!I \;\widehat{=}\; \mathbf{true} \vdash v' = v \end{array}$$

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. 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 characterised the relations that can be expressed as a design: a predicate is H1 and H2 if and only if it is a design.

H1 $R = (ok \Rightarrow R)$	No predictions before startup
H2 $[R[false/ok'] \Rightarrow R[true/ok']]$	Non-termination is not required
H3 $R = R \; ; I\!I$	Preconditions do not use dashes
H4 $R \; ; \mathbf{true} = \mathbf{true}$	Feasibility

Table 1. UTP Healthiness conditions

The healthiness conditions H3 and H4 are expressed as equations between programming constructs. Results presented in [13] 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.

A more detailed introductory account of the UTP's relational theory and designs can be found in [23]. In the next section, we consider an alternative set-based model for them.

3 Set-based model

The set-based relational model that we propose is a set of pairs of states. A state associates names (of observational variables) to their values. The sets *Name*, of valid variable names, and *Value*, of all possible values, are left unspecified. For an alphabet A, we define the set S_A of all states on A as the set of records with a component for each variable in A. Each such state can be regarded as an observation of the behaviour of a program in which the value of all the variables of the alphabet is considered.

A relation, like a UTP predicate, is a pair $(\alpha R, R)$, where αR is the alphabet of the relation, and R is a relation between the elements of $S_{in\alpha R}$ and $S_{out\alpha R}$. Such a relation models a program by associating an observation of an initial state with an observation of a possible final state.

For example, the model for an assignment $x := 3$ with alphabet $\{ x, y, x', y' \}$ is the set below.

$$\{ s : S_{\{ x,y \}};\ s' : S_{\{ x',y' \}} \mid s'.x' = 3 \wedge s'.y' = s.y \} \tag{1}$$

We use the "." to denote component selection, and also function application later on. The model for abort is the universal relation: $\mathbb{P}\, S_{in\alpha} \times S_{out\alpha}$; we use $in\alpha$ and $out\alpha$ to denote alphabets of undashed and dashed names, and omit the predicate P and the relation R when it is not relevant.

Partiality is used to model miracles. If a particular initial state is not in the domain of the relation, then it is miraculous at that state: it can achieve any required result. In particular, the model of miracle, which is the predicate **false** in the UTP setting, is the empty relation.

Standard work on relational semantics [12] singles out a special state to indicate non-termination; this is not the case in our model. 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.

In summary, the above relational model is not elaborate enough to capture non-termination. We prove in the sequel that the first general predicate-based relational model of the UTP is isomorphic to our set-based model. It is an immediate consequence of this result, that the UTP model is also not able to capture non-termination.

We define a pair of functions $p2sb$ and $sb2p$. The first transforms a UTP relation into a set-based relation; the second is its inverse: it transforms a set-based relation into a UTP relation.

Definition 1.

$$p2sb.(\alpha P, P) \;\widehat{=}\; (\alpha P, \{\, s : S_{in\alpha P};\ s' : S_{out\alpha P} \mid P[s, s'/in\alpha P, out\alpha P] \,\})$$

$$\begin{aligned}sb2p.(\alpha R, R) \;\widehat{=}\; (\alpha R, \exists\, s : S_{in\alpha R}, s' : S_{out\alpha R} \bullet (s, s') \in R \,\wedge \\ (\textstyle\bigwedge x : in\alpha R \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha R \bullet x = s'.x))\end{aligned}$$

Both $p2sb$ and $sb2p$ do not change the alphabet of the relations.

The set-based relation defined by $p2sb$ 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. For example, $p2sb.(\{\, x, y, x', y' \,\}, x := 3)$ is the pair formed by the alphabet $\{\, x, y, x', y' \,\}$ itself and the relation in (1).

The predicate defined by $sb2p$ for a relation R is an existential quantification over pairs of states s and s' in R. For each such pair, we have a conjunction of equalities that require that each observational variable takes the value mapped in the corresponding initial or final state. Since alphabets are finite, the conjunction is finite as well. As an example, we consider the application of $sb2p$ to the pair formed by $\{\, x, y, x', y' \,\}$ and the relation in (1). The result is a pair formed by the same alphabet and the predicate below.

$$\begin{aligned}
&\exists\, s : S_{\{\, x,y \,\}}, s' : S_{\{\, x',y' \,\}} \bullet \\
&\quad (s, s') \in \{\, s : S_{\{\, x,y \,\}};\ s' : S_{\{\, x',y' \,\}} \mid s'.x' = 3 \wedge s'.y' = s.y \,\} \,\wedge \\
&\quad x = s.x \wedge y = s.y \wedge x' = s'.x' \wedge y' = s'.y' \\[1ex]
&= \exists\, s : S_{\{\, x,y \,\}}, s' : S_{\{\, x',y' \,\}} \bullet && \text{[property of sets]} \\
&\qquad s'.x' = 3 \wedge s'.y' = s.y \ \wedge x = s.x \wedge y = s.y \wedge x' = s'.x' \wedge y' = s'.y' \\[1ex]
&= \exists\, s : S_{\{\, x,y \,\}}, s' : S_{\{\, x',y' \,\}} \bullet && \text{[property of equality]} \\
&\qquad s'.x' = 3 \wedge s'.y' = s.y \ \wedge x = s.x \wedge y = s.y \wedge x' = 3 \wedge y' = y \\[1ex]
&= (\exists\, s : S_{\{\, x,y \,\}}, s' : S_{\{\, x',y' \,\}} \bullet s'.x' = 3 \wedge s'.y' = s.y \ \wedge x = s.x \wedge y = s.y) \,\wedge \\
&\quad x' = 3 \wedge y' = y && \text{[predicate calculus]} \\[1ex]
&= (x' = 3 \wedge y' = y) && \text{[equality of records and one-point rule]} \\[1ex]
&= (x := 3) && \text{[UTP general model of assignment]}
\end{aligned}$$

This is the result to be expected, since we have already pointed out that the relation in (1) is the model for the assignment $x := 3$.

The theorem below shows that $p2sb$ and $sb2p$ establish an isomorphism between the UTP predicative relations and our set-based relational model.

Theorem 1. *For a set-based relation $(\alpha R, R)$, and a UTP relation $(\alpha P, P)$, we have $p2sb.(sb2p.(\alpha R, R)) = (\alpha R, R)$ and $sb2p.(p2sb.(\alpha P, P)) = (\alpha P, P)$.*

Proof. The alphabets are maintained by both $p2sb$ and $sb2p$, so they are largely ignored in the proof below, where we regard these functions as acting on predicates and on sets of pairs of states, instead of on pairs. We also omit the types of state variables s and s' as they can be inferred from the context. We adopt the same approach in subsequent proofs.

First, we consider the relation $p2sb.(sb2p.R)$.

$$\begin{aligned}
& p2sb.(sb2p.R) \\
&= \{\, s, s' \mid (sb2p.R)[s, s'/in\alpha, out\alpha] \,\} && \text{[definition of } p2sb\text{]} \\
&= \{\, s, s' \mid (\exists\, s, s' \bullet (s, s') \in R \wedge && \text{[definition of } sb2p\text{]} \\
&\qquad\qquad (\textstyle\bigwedge x : in\alpha P \bullet x = s.x) \wedge \\
&\qquad\qquad (\textstyle\bigwedge x : out\alpha \bullet x = s'.x))[s, s'/in\alpha, out\alpha] \,\} \\
&= \{\, s, s' \mid \exists\, s_1, s_2 \bullet (s_1, s_2) \in R \wedge && \text{[property of substitution]} \\
&\qquad\qquad (\textstyle\bigwedge x : in\alpha \bullet s.x = s_1.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet s'.x = s_2.x) \,\} \\
&= \{\, s, s' \mid \exists\, s_1, s_2 \bullet (s_1, s_2) \in R \wedge s = s_1 \wedge s' = s_2 \,\} && \text{[equality of records]} \\
&= \{\, s, s' \mid (s, s') \in R \,\} && \text{[one-point rule]} \\
&= R && \text{[property of sets]}
\end{aligned}$$

Now, we consider the predicate $sb2p.(p2sb.P)$. In the proof below, we use $P[A/s]$ to denote the substitution in P of x for $s.x$, for all x in A.

$$\begin{aligned}
& sb2p.(p2sb.P) \\
&= \exists\, s, s' \bullet (s, s') \in p2sb.P \wedge (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x) \\
& && \text{[definition of } sb2p\text{]} \\
&= \exists\, s, s' \bullet (s, s') \in \{\, s, s' \mid P[s, s'/in\alpha P, out\alpha] \,\} \wedge && \text{[definition of } p2sb\text{]} \\
&\qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x) \\
&= \exists\, s, s' \bullet P[s, s'/in\alpha, out\alpha] \wedge && \text{[property of sets]} \\
&\quad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x) \\
&= P[s, s'/in\alpha, out\alpha][in\alpha, out\alpha/s, s'] && \text{[equality of records and one-point rule]} \\
&= P && \text{[} s \text{ and } s' \text{ are fresh names and property of substitution]}
\end{aligned}$$

□

This theorem confirms that the general UTP model for relations is not able to capture non-termination. This may not seem very obvious from the predicate-based model, but it is a straightforward observation in the context of the set-based model. What we have is a model of terminating programs.

Hoare and He pointed out a paradox in the fact that, if the alphabet is $\{\, x, x' \,\}$, then $(\mu\, X \bullet X);\ x := 3$ is equivalent to $x := 3$. This is not really a paradox: the bottom of the lattice $\bot$ is not really an aborting program; it is the program that terminates, but gives an arbitrary value to x. If, in sequence, we assign 3 to x, then the arbitrariness of $(\mu X \bullet X)$ is irrelevant. Their model is quite 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. The model is simply not elaborate enough. By the way, in that study, they seem ready to accept the law $\top = P \mathbin{;} \top$, which states that, if we follow any program by a miracle, then we get a miracle. This is only acceptable if we are indeed dealing with terminating programs. If P does not terminate, then $P \mathbin{;} \top$ is never going to be miraculous. This is actually a law of the simple relational model that we do not want to preserve in a model that handles non-terminating programs.

Our main point, at this stage, is that the study of the set-based model can be insightful. We, therefore, proceed by considering the set-based relations that correspond to designs, the more elaborate relations that Hoare and He accept as the basis for their further work.

The alphabet of designs includes the variables ok and ok'; therefore, these variables are also part of the alphabet of the corresponding set-based relations. Moreover, in Table 1, we have presented healthiness conditions that have been considered of interest by Hoare and He. In Table 2, we present corresponding healthiness conditions for our set-based relations R.

SBH1	$\forall s, s' \mid s.ok = \mathit{false} \bullet (s, s') \in R$
SBH2	$\forall s, s' \mid (s, s') \in R \land s'.ok' = \mathit{false} \bullet (s, s' \oplus \{ok' \mapsto \mathit{true}\}) \in R$
SBH3	$\forall s \mid (\exists s' \bullet s'.ok' = \mathit{false} \land (s, s') \in R) \bullet \forall s' \bullet (s, s') \in R$

Table 2. Set-based healthiness conditions

The healthiness condition SBH1 requires that all initial 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 an initial state s is related to a final state s' for which $s'.ok'$ is false, then s is also related to $s' \oplus \{ok' \mapsto \mathit{true}\}$. This is the same state as s', except only 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. It is interesting to note, however, that even if it is possible for R not to terminate, its behaviour may not be completely arbitrary, since it is not required that R relates s to all possible final states. This is what is required by SBH3.

The theorems below establish that H1 designs correspond to SBH1 relations, and H2 designs correspond to SBH2 relations. A consequence of these results is that SBH1 and SBH2 characterise a set-based theory of designs.

Theorem 2. *For every UTP relation $(\alpha P, P)$ that satisfies* H1*, $p2sb.(\alpha P, P)$ satisfies* SBH1*. Conversely, for every set-based relation $(\alpha R, R)$ that satisfies* SBH1*, $sb2p.(\alpha R, R)$ satisfies* H1*.*

Proof. First, we consider a predicate P that satisfies H1.

$$\begin{aligned}
& p2sb.P \\
& = \{\, s, s' \mid P[s, s'/in\alpha, out\alpha] \,\} && [\text{definition of } p2sb] \\
& = \{\, s, s' \mid (ok \Rightarrow P)[s, s'/in\alpha, out\alpha] \,\} && [\mathsf{H1}] \\
& = \{\, s, s' \mid s.ok = false \vee P[s, s'/in\alpha, out\alpha] \,\} \\
& && [\text{predicate calculus and substitution}] \\
& = \{\, s, s' \mid s.ok = false \,\} \cup \{\, s, s' \mid P[s, s'/in\alpha P, out\alpha] \,\} && [\text{property of sets}] \\
& = \{\, s, s' \mid s.ok = false \,\} \cup p2sb.P && [\text{definition of } p2sb]
\end{aligned}$$

Now, we consider a SBH1-healthy relation.

$$\begin{aligned}
& sb2p.R \\
& = \exists\, s, s' \bullet (s, s') \in R \wedge (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x) \\
& && [\text{definition of } sb2p.R] \\
& = \exists\, s, s' \bullet (s, s') \in \{\, s, s' \mid s.ok = false \,\} \cup R \wedge && [\mathsf{SBH1}] \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x) \\
& = (\exists\, s, s' \bullet (s, s') \in \{\, s, s' \mid s.ok = false \,\} \wedge \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x)) \vee \\
& \quad (\exists\, s, s' \bullet (s, s') \in R \wedge (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x)) \\
& && [\text{property of sets and predicate calculus}] \\
& = (\exists\, s, s' \bullet (s, s') \in \{\, s, s' \mid s.ok = false \,\} \wedge && [\text{definition of } sb2p] \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x)) \vee \\
& \quad sb2p.R \\
& = (\exists\, s, s' \bullet s.ok = false \wedge (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x)) \vee \\
& \quad sb2p.R && [\text{property of sets}] \\
& = (\exists\, s, s' \bullet \neg\, ok \wedge (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \wedge (\textstyle\bigwedge x : out\alpha \bullet x = s'.x)) \vee sb2p.R \\
& && [ok \in in\alpha \text{ and property of equality}] \\
& = \neg\, ok \vee sb2p.R && [\text{equality of records and one-point rule}] \\
& = ok \Rightarrow sb2p.R && [\text{predicate calculus}]
\end{aligned}$$

□

Theorem 3. *For every UTP relation $(\alpha P, P)$ that satisfies* H2*, $p2sb.(\alpha P, P)$ satisfies* SBH2*. Conversely, for every set-based relation $(\alpha R, R)$ that satisfies* SBH2*, $sb2p.(\alpha R, R)$ satisfies* H2*.*

Proof. We first consider a predicate P that satisfies H2, and states s and s' such that $s'.ok' = \mathit{false}$.

$$
\begin{aligned}
& (s, s') \in p2sb.P \\
& \Rightarrow P[s, s'/in\alpha, out\alpha] && \text{[definition of } p2sb\text{]} \\
& = P[s'.ok'/ok'][s, s'/in\alpha, out\alpha] && \text{[property of substitution]} \\
& = P[\mathit{false}/ok'][s, s'/in\alpha, out\alpha] && \text{[assumption]} \\
& \Rightarrow P[\mathit{true}/ok'][s, s'/in\alpha, out\alpha] && \text{[H2]} \\
& = P[s, s' \oplus \{\, ok' \mapsto \mathit{true} \,\}/in\alpha, out\alpha] && \text{[property of substitution]} \\
& \Rightarrow (s, s' \oplus \{\, ok' \mapsto \mathit{true} \,\}) \in p2sb.P && \text{[definition of } p2sb\text{]}
\end{aligned}
$$

Secondly, we consider an SBH2-healthy relation R.

$$
\begin{aligned}
& (sb2p.R)[\mathit{false}/ok'] \\
& = (\exists\, s, s' \bullet (s, s') \in R \land && \text{[definition of } sb2p.R\text{]} \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \land (\textstyle\bigwedge x : out\alpha \bullet x = s'.x))[\mathit{false}/ok'] \\
& = \exists\, s, s' \bullet (s, s') \in R \land && \text{[property of substitution]} \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \land (\textstyle\bigwedge x : out\alpha \setminus \{\, ok' \,\} \bullet x = s'.x) \land s'.ok' = \mathit{false} \\
& \Rightarrow \exists\, s, s' \bullet (s, s' \oplus \{\, ok' \mapsto \mathit{true} \,\}) \in R \land && \text{[SBH2 and predicate calculus]} \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \land (\textstyle\bigwedge x : out\alpha \setminus \{\, ok' \,\} \bullet x = s'.x) \\
& = \exists\, s, s' \bullet (s, s' \oplus \{\, ok' \mapsto \mathit{true} \,\}) \in R \land && \text{[properties of records]} \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \land \\
& \qquad (\textstyle\bigwedge x : out\alpha \setminus \{\, ok' \,\} \bullet x = (s' \oplus \{\, ok' \mapsto \mathit{true} \,\}).x) \land \\
& \qquad (s' \oplus \{\, ok' \mapsto \mathit{true} \,\}).ok' = \mathit{true} \\
& \Rightarrow \exists\, s, s' \bullet (s, s') \in R \land && \text{[predicate calculus]} \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \land (\textstyle\bigwedge x : out\alpha \setminus \{\, ok' \,\} \bullet x = s'.x) \land s'.ok' = \mathit{true} \\
& = (\exists\, s, s' \bullet (s, s') \in R \land && \text{[property of substitution]} \\
& \qquad (\textstyle\bigwedge x : in\alpha \bullet x = s.x) \land (\textstyle\bigwedge x : out\alpha \bullet x = s'.x))[\mathit{true}/ok'] \\
& = (sb2p.R)[\mathit{true}/ok'] && \text{[definition of } sb2p\text{]}
\end{aligned}
$$

□

We observe that Hoare and He point out that non-H3 designs include designs whose preconditions include dashed observational variables. As such, it is not possible for a user of the program to guarantee that its precondition is going to be satisfied. We give the design $x' \neq 2 \vdash \mathbf{true}$ as an example; it is equivalent to $ok \Rightarrow x' = 2 \vee ok'$. This is a program that, if started, then it may either terminate and assign an arbitrary value to x, or set the value of x to 2, in which case it is not required to terminate. The user cannot prevent the program from setting x to 2 and failing to terminate; this is a rather bizarre situation.

Consequently, Hoare and He imply that we should work with H3 designs. The healthiness condition SBH3 expresses the relevant restriction in the set-based model. It requires that, if an initial state s is related to a non-terminating state s', then it is related to all possible states. The theorem below formalises the correspondence with H3.

Theorem 4. *For every UTP relation $(\alpha P, P)$ that satisfies H3, $p2sb.(\alpha P, P)$ satisfies SBH3. Conversely, for every set-based relation $(\alpha R, R)$ that satisfies SBH3, $sb2p.(\alpha R, R)$ satisfies H3.*

Proof. In the proof that $p2sb.P$ is SBH3-healthy, we assume that the alphabet is composed of variables v, v', ok and ok'.

$$p2sb.P(v', ok')$$

$$= \{ s, s' \mid P(v', ok')[s, s'/in\alpha, out\alpha] \} \qquad \text{[definition of } p2sb\text{]}$$

$$= \{ s, s' \mid (P(v', ok');\ I\!I)[s, s'/in\alpha, out\alpha] \} \qquad \text{[H3]}$$

$$= \{ s, s' \mid (P(v', ok');\ (ok \Rightarrow v' = v \wedge ok'))[s, s'/in\alpha, out\alpha] \}$$
$$\text{[definition of } I\!I\text{]}$$

$$= \{ s, s' \mid \exists\, v_0, ok_0 \bullet P(v_0, ok_0)[s/in\alpha] \wedge (ok_0 \Rightarrow s'.v' = v_0 \wedge s'.ok' = true) \}$$
$$\text{[definition of ; and property of substitution]}$$

$$= \{ s, s' \mid \exists\, v_0, ok_0 \bullet P(v_0, ok_0)[s/in\alpha] \wedge \qquad \text{[property of equality]}$$
$$(ok_0 \Rightarrow s'.v' = v_0 \wedge ok_0 = s'.ok' \wedge s'.ok' = true) \}$$

$$= \{ s, s' \mid (\exists\, v_0, ok_0 \bullet P(v_0, ok_0)[s/in\alpha] \wedge \neg\, ok_0) \vee \qquad \text{[predicate calculus]}$$
$$(\exists\, v_0, ok_0 \bullet P(v_0, ok_0)[s/in\alpha] \wedge$$
$$s'.v' = v_0 \wedge ok_0 = s'.ok' \wedge s'.ok' = true) \}$$

$$= \{ s, s' \mid (\exists\, v_0, ok_0 \bullet P(v_0, ok_0)[s/in\alpha] \wedge \neg\, ok_0) \vee \qquad \text{[one-point rule]}$$
$$P(v_0, ok_0)[s, s'/in\alpha, \{ v_0, ok_0 \}] \wedge s'.ok' = true \}$$

$$= \{ s, s' \mid (\exists\, v_0, ok_0 \bullet P(v_0, ok_0)[s/in\alpha] \wedge \neg\, ok_0) \vee$$
$$P[s, s'/in\alpha, out\alpha] \wedge s'.ok' = true \}$$
$$[v_0 \text{ and } ok_0 \text{ are fresh and property of substitution}]$$

$$= \{ s, s' \mid (\exists s' \bullet P[s, s'/in\alpha, out\alpha] \land s'.ok' = \mathit{false}) \lor P[s, s'/in\alpha, out\alpha] \land s'.ok' = \mathit{true} \}$$

[equality of records and predicate calculus]

$$= \{ s, s' \mid \exists s' \bullet P[s, s'/in\alpha, out\alpha] \land s'.ok' = \mathit{false} \} \cup \{ s, s' \mid P[s, s'/in\alpha, out\alpha] \land s'.ok' = \mathit{true} \}$$ [property of sets]

$$= \{ s, s' \mid \exists s' \bullet (s, s') \in p2sb.P \land s'.ok' = \mathit{false} \} \cup \{ s, s' \mid (s, s') \in p2sb.P \land s'.ok' = \mathit{true} \}$$ [definition of $p2sb$]

$(sb2p.R);\ I\!I$

$$= (\exists s, s' \bullet (s, s') \in R \land (\bigwedge x : in\alpha \bullet x = s.x) \land (\bigwedge x' : out\alpha \bullet x' = s'.x')); I\!I$$ [definition of $sb2p$]

$$= (\exists s, s' \bullet (s, s') \in R \land (\bigwedge x : in\alpha \bullet x = s.x) \land (\bigwedge x' : out\alpha \bullet x' = s'.x')); (ok \Rightarrow (\bigwedge x' : out\alpha \bullet x' = x) \land ok')$$ [definition of $I\!I$]

$$= \exists s, s' \bullet (s, s') \in R \land (\bigwedge x : in\alpha \bullet x = s.x) \land (s'.ok' = \mathit{true} \Rightarrow (\bigwedge x' : out\alpha \bullet x' = s'.x') \land ok')$$

[definition of ; and one-point rule]

$$= \exists s, s' \bullet (s, s') \in R \land (\bigwedge x : in\alpha \bullet x = s.x) \land (s'.ok' = \mathit{true} \Rightarrow (\bigwedge x' : out\alpha \bullet x' = s'.x') \land s'.ok' = \mathit{true})$$ [property of equality]

$$= \exists s, s' \bullet (s, s') \in R \land (\bigwedge x : in\alpha \bullet x = s.x) \land (s'.ok' = \mathit{false} \lor (\bigwedge x' : out\alpha \bullet x' = s'.x'))$$ [predicate calculus]

$$= (\exists s, s' \bullet (s, s') \in R \land (\bigwedge x : in\alpha \bullet x = s.x) \land s'.ok' = \mathit{false}) \lor (\exists s, s' \bullet (s, s') \in R \land (\bigwedge x : in\alpha \bullet x = s.x) \land (\bigwedge x' : out\alpha \bullet x' = s'.x'))$$

[predicate calculus]

$$= \exists s, s' \bullet (s, s') \in R \land (\bigwedge x : in\alpha \bullet x = s.x) \land (\bigwedge x' : out\alpha \bullet x' = s'.x'))$$

[SBH3 and predicate calculus]

The details of the justification of the last step are as follows; we prove that the second disjunct follows from the first.

$$\exists s, s' \bullet (s, s') \in R \land s'.ok' = \mathit{false}$$

$$\Rightarrow \exists s \bullet \forall s' \bullet (s, s') \in R$$ [SBH3]

$$\Rightarrow \exists s, s' \bullet (\bigwedge x' : out\alpha \bullet s'.x' = x') \land (s, s') \in R$$

[predicate calculus, equality of records, and one-point rule]

We can finally conclude the proof as follows.

$$\exists\, s, s' \bullet (s, s') \in R \wedge (\bigwedge x : in\alpha \bullet x = s.x) \wedge (\bigwedge x' : out\alpha \bullet x' = s'.x'))$$

$$= sb2p.R \qquad \text{[definition of } sb2p\text{]}$$

□

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 $s' \oplus \{ok' \mapsto true\}$. 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 condition, but only three of them: H1, H3, and H4. Out of curiosity, we present below, in the setting of UTP relations, a proof of the fact that H3 designs are also H2.

Theorem 5. *If a UTP relation $(\alpha P, P)$ satisfies H3, then it also satisfies H2.*

Proof.

$$
\begin{aligned}
& P(v', ok')[false/ok'] \\
&= (P(v', ok') \,;\, I\!I)[false/ok'] && \text{[H3]} \\
&= (\exists\, v_0, ok_0 \bullet P(v_0, ok_0) \wedge (ok_0 \Rightarrow v_0 = v' \wedge ok'))[false/ok'] \\
& && \text{[definitions of } I\!I \text{ and ;]} \\
&= \exists\, v_0, ok_0 \bullet P(v_0, ok_0) \wedge (ok_0 \Rightarrow v_0 = v' \wedge false)) && \text{[property of substitution]} \\
&= \exists\, v_0, ok_0 \bullet P(v_0, ok_0) \wedge \neg\, ok_0 && \text{[predicate calculus]} \\
&\Rightarrow (\exists\, v_0, ok_0 \bullet P(v_0, ok_0) \wedge \neg\, ok_0) \vee (\exists\, v_0, ok_0 \bullet P(v_0, ok_0) \wedge v_0 = v') \\
& && \text{[predicate calculus]} \\
&= \exists\, v_0, ok_0 \bullet P(v_0, ok_0) \wedge (\neg\, ok_0 \vee v_0 = v') && \text{[predicate calculus]} \\
&= \exists\, v_0, ok_0 \bullet P(v_0, ok_0) \wedge (ok_0 \Rightarrow v_0 = v') && \text{[predicate calculus]} \\
&= (P(v', ok') \,;\, I\!I)[true/ok'] && \text{[definition of ; and } I\!I\text{]} \\
&= P(v', ok')[true/ok'] && \text{[H3]}
\end{aligned}
$$

□

It may well be the case that non-H3 designs are important for the modelling of more sophisticated programming paradigms that include sequential programs. We leave this for further investigation.

The healthiness condition H4 requires feasibility. It is not of paramount concern for us, as miracles are quite an important part of Morgan's refinement calculus and ZRC. We, therefore, ignore it in our study. In the next section, we explore an alternative model for UTP relations and designs.

4 Predicate transformers

In the model of predicate transformers, we regard predicates as sets of states. The model is composed of pairs $(\alpha PT, PT)$, where αPT is the alphabet of the transformer, and PT is a total monotonic function from $\mathbb{P}\, S_{out\alpha PT}$ to $\mathbb{P}\, S_{in\alpha PT}$. A program is modelled by its weakest precondition transformer [8].

Isomorphisms between predicate transformers and set-based relational models have already been studied [11]. The isomorphism that we propose here is similar to that in [6]. We define functions *sb2pt* and *pt2sb*; 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 2.

$$sb2pt.R.\psi \mathrel{\widehat{=}} \overline{\operatorname{dom}(R \rhd \psi)}$$

$$pt2sb.PT = \{\, s : S_{in\alpha PT};\ s' : S_{out\alpha PT} \mid s \in \overline{PT.\overline{\{\, s' \,\}}} \,\}$$

In the definition of *sb2pt*, ψ is a postcondition, or rather, a set of states, which is given as argument to the transformer $sb2pt.R$. The relation $R \rhd \psi$ models all executions of R that do not lead to a final state that satisfies ψ. In $\operatorname{dom}(R \rhd \psi)$, 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.

The relation $pt2sb.PT$ 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, we prove that they are isomorphic to the set of universally conjunctive predicate transformers PT: those that satisfy the property below.

$$PT.(\bigcap\{\, i \bullet \psi_i \,\}) = \bigcap\{\, i \bullet PT.\psi_i \,\} \tag{2}$$

The following theorems establish this result. First of all, we have that the range of *sb2pt* is the set of universally conjunctive predicate transformers.

Theorem 6.

$$sb2pt.R.(\bigcap\{\, i \bullet \psi_i \,\}) = \bigcap\{\, i \bullet sb2pt.R.\psi_i \,\}$$

The proof of this result is similar to that of a corresponding result in [6].

Now, we have the theorem that establishes the isomorphism.

Theorem 7. *For a set-based relation R, and a universally conjunctive predicate transformer PT, we have $sb2pt.(pt2sb.PT) = PT$, and $pt2sb.(sb2pt.R) = R$.*

Proof. A result similar to $pt2sb.(sb2pt.R) = R$ is proved in [6]. The proof of $sb2pt.(pt2sb.PT) = PT$ requires universal conjunctivity and is presented below.

$$
\begin{array}{ll}
sb2pt.(pt2sb.PT).\psi & \\
= \overline{\mathrm{dom}(pt2sb.PT \rhd \psi)} & [\text{definition of } sb2pt] \\
= \overline{\mathrm{dom}(\{\, s, s' \mid s \in \overline{PT.\overline{\{\, s' \,\}}} \,\} \rhd \psi)} & [\text{definition of } pt2sb] \\
= \overline{\{\, s \mid \exists\, s' : \overline{\psi} \bullet s \in \overline{PT.\overline{\{\, s' \,\}}} \,\}} & [\text{properties of } \rhd \text{ and dom}] \\
= \{\, s \mid \forall\, s' : \overline{\psi} \bullet s \in PT.\overline{\{\, s' \,\}} \,\} & [\text{properties of sets and predicate calculus}] \\
= \{\, s \mid s \in \bigcap\{\, s' : \overline{\psi} \bullet PT.\overline{\{\, s' \,\}} \,\} \,\} & [\text{property of set intersection}] \\
= \{\, s \mid s \in PT.\bigcap\{\, s' : \overline{\psi} \bullet \overline{\{\, s' \,\}} \,\} \,\} & [\text{universal conjunctivity}] \\
= \{\, s \mid s \in PT.\psi \,\} & [\text{property of sets}] \\
= PT.\psi & [\text{property of sets}]
\end{array}
$$

□

Hoare and He define a weakest precondition operator, **wp**, for the UTP relations. It is reassuring that it corresponds to our characterisation of weakest preconditions.

For a relation Q, and a condition r, which is a relation whose alphabet does not include dashed variables, Hoare and He have the following definition.

$$Q \ \mathbf{wp}\ r \mathrel{\widehat=} \neg\,(Q;\ \neg\, r)$$

The alphabet of r must be such that $out\alpha Q = in\alpha' r$.

To establish the relationship between **wp** and our predicate transformers, we define functions $c2sb$ and $sb2c$, which are similar to $p2sb$ and $sb2p$; the difference is that they act on sets of states, instead of on relations on states.

Definition 3.

$$
\begin{array}{l}
c2sb.C \mathrel{\widehat=} \{\, s : S_{in\alpha C} \mid C[s/\alpha C] \,\} \\
sb2c.SS \mathrel{\widehat=} \exists\, s : SS \bullet s \in SS \wedge (\bigwedge x : \alpha SS \bullet x = s.x)
\end{array}
$$

The fact that $c2sb$ and $sb2c$ establish an isomorphism between sets of states and UTP conditions can be proved in much the same way as we proved that $p2sb$ and $sb2p$ establish an isomorphism between set-based and UTP relations. It is an advantage of the predicate-based model that conditions and relations are handled seamlessly.

Using $c2sb$ and $sb2c$, we can establish the following result about **wp**, where r' is the predicate that is obtained by dashing the free variables of r. This is a postcondition, as its alphabet contains only dashed variables.

Theorem 8.

$$Q \ \mathbf{wp} \ r = sb2c.(sb2pt.(p2sb.Q).(c2sb.r'))$$

Proof.

$$\begin{aligned}
& sb2pt.(p2sb.Q).(c2sb.r') \\
&= \overline{\mathrm{dom}(p2sb.Q) \rhd c2sb.r'} && [\text{definition of } sb2pt] \\
&= \overline{\mathrm{dom}\{\, s, s' \mid Q[s, s'/in\alpha Q, out\alpha Q]\,\} \rhd c2sb.r'} && [\text{definition of } p2sb] \\
&= \overline{\mathrm{dom}\{\, s, s' \mid Q[s, s'/in\alpha Q, out\alpha Q]\,\} \rhd \{\, s' \mid r'[s'/out\alpha Q]\,\}} \\
& && [\text{definition of } c2sb] \\
&= \overline{\mathrm{dom}\{\, s, s' \mid Q[s, s'/in\alpha Q, out\alpha Q] \land \neg\, r'[s'/out\alpha Q]\,\}} && [\text{property of } \rhd] \\
&= \overline{\{\, s \mid \exists\, s' \bullet Q[s, s'/in\alpha Q, out\alpha Q] \land \neg\, r'[s'/out\alpha Q]\,\}} && [\text{definition of dom}] \\
&= \{\, s \mid \neg\, \exists\, s' \bullet Q[s, s'/in\alpha Q, out\alpha Q] \land \neg\, r'[s'/out\alpha Q]\,\} && [\text{property of sets}] \\
&= \{\, s \mid \neg\, (\exists\, s' \bullet Q[s'/out\alpha Q] \land \neg\, r[s'/in\alpha Q])[s/in\alpha Q]\,\} \\
& && [\text{property of substitution}] \\
&= \{\, s \mid \neg\, (\exists\, out\alpha_0 Q \bullet Q[out\alpha_0 Q/out\alpha Q] \land \neg\, r[out\alpha_0 Q/in\alpha Q])[s/in\alpha Q]\,\} \\
& && [\text{predicate calculus}] \\
&= \{\, s \mid \neg\, (Q;\ \neg\, r)[s/in\alpha Q]\,\} && [\text{definition of sequence}] \\
&= \{\, s \mid (Q \ \mathbf{wp} \ r)[s/in\alpha Q]\,\} && [\text{definition of } \mathbf{wp}] \\
&= c2sb.(Q \ \mathbf{wp} \ r) && [\text{definition of } c2sb]
\end{aligned}$$

□

The above theorem states that, in order to calculate Q **wp** r, we can transform Q into a set-based relation, and then into a predicate transformer, apply the result to the set of states corresponding to r', and finally transform the resulting weakest precondition into a condition. For conciseness, we omit the proof.

For our objectives the most important consequence of Theorem 7 is that UTP relations cannot model angelic nondeterminism. Since we have an isomorphism between UTP relations and set-based relations, and another between set-based relations and universally conjunctive predicate transformers, then UTP relations are isomorphic to universally conjunctive predicate transformers.

As already mentioned, the angelic choice operator in which we are interested is the least upper bound of the lattice of monotonic predicate transformers. In [3], Back and von Wright establish that joins in the lattice of universally conjunctive predicate transformers are not preserved in the lattice of monotonic predicate transformers. We need a relational model isomorphic to the monotonic predicate transformers. We investigate, next, the set of predicate transformers that correspond to UTP designs.

In this case, ok is in the alphabet of the states in a precondition, and ok' is in the alphabet of the states in a postcondition. Table 3 gives healthiness conditions over such predicate transformers PT. The first healthiness condition, PTH1 requires that the weakest precondition for PT to establish 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 $S_{out\alpha PT}$, which imposes no restrictions whatsoever.

PTH1	$PT.\psi \subseteq \{s : S_{in\alpha PT} \mid s.ok = true\}$ provided $\psi \neq S_{out\alpha PT}$
PTH2	$PT.\psi = PT.\{s' : \psi \mid s' \oplus \{ok' \mapsto true\} \in \psi\}$
PTH3	$PT.\psi = PT.\{s' : \psi \mid s'.ok' = true\}$ provided $\psi \neq S_{out\alpha PT}$

Table 3. Predicate transformers healthiness conditions

The healthiness condition PTH2 states that we can calculate $PT.\psi$ by considering the subset of ψ formed by the states s' such that either $s'.ok'$ is true or $s' \oplus \{ok' \mapsto true\}$ is in the set as well. This is because, if $s'.ok'$ is false and termination, or rather, $s'.ok' = true$, is not acceptable, then PT cannot guarantee s', as it cannot guarantee non-termination. For this reason, we may as well ignore s' when determining the initial states that guarantee that one of the final states in ψ is going to be achieved. It is interesting to observe that if ψ is $S_{out\alpha PT}$, then the subset of ψ considered in PTH2 is $S_{out\alpha PT}$ itself.

Finally, the healthiness condition PTH3 states that, in calculating $PT.\psi$, we can ignore all the states s' in ψ for which $s'.ok'$ is false. In other words, even if we have both s' and $s' \oplus \{ok' \mapsto true\}$ are in ψ, so that termination is not required, if PT can guarantee either s' or $s' \oplus \{ok' \mapsto true\}$, then it can guarantee $s' \oplus \{ok' \mapsto true\}$. This is because the healthy predicate transformers do not capture any relevant information if there is a possibility of non-termination. Again, the postcondition $S_{out\alpha PT}$ is an exception.

As expected, PTH1, PTH2, and PTH3 correspond to H1, H2, and H3. We prove this result below, using SBH1, SBH2, and SBH3.

Theorem 9. *For every set-based relation R that satisfies SBH1, $sb2pt.R$ satisfies PTH1. Conversely, for every predicate transformer PT that satisfies PTH1, $pt2sb.PT$ satisfies SBH1.*

Proof. We consider a postcondition $\psi \neq S_{out\alpha R}$.

$$
\begin{array}{ll}
sb2pt.R.\psi \cap \{\, s \mid s.ok = true \,\} & \\
= \overline{\mathrm{dom}(R \rhd \overline{\psi})} \cap \{\, s \mid s.ok = true \,\} & [\text{definition of } sb2pt] \\
= \{\, s \mid \forall\, s' \bullet (s, s') \in R \Rightarrow s' \in \psi \,\} \cap \{\, s \mid s.ok = true \,\} & \\
\qquad [\text{properties of } \rhd\text{, dom and sets, and predicate calculus}] & \\
= \{\, s \mid (\forall\, s' \bullet (s, s') \in R \Rightarrow s' \in \psi) \wedge s.ok = true \,\} & [\text{property of sets}]
\end{array}
$$

From the assumption, we can conclude that the second conjunct in this set comprehension follows from the first; we prove this below.

$$
\begin{array}{ll}
(\forall\, s' \bullet (s, s') \in R \Rightarrow s' \in \psi) \wedge \psi \neq S_{out\alpha R} & \\
= (\forall\, s' \bullet (s, s') \in R \Rightarrow s' \in \psi) \wedge (\exists\, s' \bullet s' \notin \psi) & [\psi \subseteq S_{out\alpha R}] \\
= (\forall\, s' \bullet s' \notin \psi \Rightarrow (s, s') \notin R) \wedge (\exists\, s' \bullet s' \notin \psi) & [\text{predicate calculus}] \\
\Rightarrow \exists\, s' \bullet (s, s') \notin R & [\text{predicate calculus}] \\
\Rightarrow \exists\, s' \bullet s.ok \neq false & [\mathsf{SBH1} \text{ and contraposition}] \\
= (s.ok = true) & [s.ok \text{ is boolean}]
\end{array}
$$

And so we reach our result.

$$
\begin{array}{ll}
\{\, s : S_{in\alpha R} \mid (\forall\, s' \bullet (s, s') \in R \Rightarrow s' \in \psi) \wedge s.ok = true \,\} & \\
= \{\, s \mid \forall\, s' \bullet (s, s') \in R \Rightarrow s' \in \psi \,\} & [\text{from above}] \\
= sb2pt.R.\psi & [\text{definition of } sb2pt]
\end{array}
$$

For the healthiness of $pt2sb.PT$, we have the following.

$$
\begin{array}{ll}
\{\, s, s' \mid s.ok = false \,\} \subseteq pt2sb.PT & \\
= \{\, s, s' \mid s.ok = false \,\} \subseteq \{\, s, s' \mid s \in \overline{PT.\overline{\{s'\}}} \,\} & [\text{definition of } pt2sb] \\
= \forall\, s, s' \bullet s.ok = false \Rightarrow s \in \overline{PT.\overline{\{s'\}}} & [\text{property of sets}] \\
= \forall\, s, s' \bullet s.ok = false \Rightarrow s \notin PT.\overline{\{s'\}} & [\text{property of sets}] \\
= \forall\, s, s' \bullet s.ok = false \Rightarrow s \notin PT.\overline{\{s'\}} \cap \{\, s \mid s.ok = true \,\} & [\mathsf{PTH1}] \\
= true & [\text{property of sets}]
\end{array}
$$

□

Theorem 10. *For every set-based relation R that satisfies* $\mathsf{SBH2}$*, $sb2pt.R$ satisfies* $\mathsf{PTH2}$*. Conversely, for every predicate transformer PT that satisfies* $\mathsf{PTH2}$*, $pt2sb.PT$ satisfies* $\mathsf{SBH2}$*.*

Proof. To prove that $pt2sb.PT$ is $\mathsf{SBH2}$-healthy, we consider s and s' such that $(s, s') \in pt2sb.PT$ and $s'.ok' = \mathit{false}$.

$$
\begin{aligned}
& (s, s') \in pt2sb.PT \\
& = s \in \overline{PT.\overline{\{ s' \}}} && \text{[definition of } pt2sb\text{]} \\
& = s \notin PT.\overline{\{ s' \}} && \text{[property of sets]} \\
& \Rightarrow s \notin PT.\overline{\{ s', s' \oplus \{ ok' \mapsto \mathit{true} \} \}} && \text{[monotonicity of } PT\text{]} \\
& = s \notin PT.(\overline{\{ s', s' \oplus \{ ok' \mapsto \mathit{true} \} \}} \cup s') \\
& \qquad [\mathsf{PTH2},\ s'.ok' = \mathit{false}, \text{ and } s' \oplus \{ok' \mapsto \mathit{true}\} \notin \overline{\{ s', s' \oplus \{ok' \mapsto \mathit{true}\} \}}] \\
& = s \notin PT.\overline{\{ s' \oplus \{ ok' \mapsto \mathit{true} \} \}} && \text{[property of sets]} \\
& = s \in \overline{PT.\overline{\{ s' \oplus \{ ok' \mapsto \mathit{true} \} \}}} && \text{[property of sets]} \\
& = (s, s' \oplus \{ ok' \mapsto \mathit{true} \}) \in pt2sb.PT && \text{[definition of } pt2sb\text{]}
\end{aligned}
$$

For the healthiness of $sb2pt.R$, we proceed as follows.

$$
\begin{aligned}
& sb2pt.R.\psi \\
& = \overline{\operatorname{dom}(R \rhd \psi)} && \text{[definition of } sb2pt\text{]} \\
& = \{ s \mid \forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \} && \text{[properties of } \rhd\text{, dom, and sets]} \\
& = \{ s \mid \forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \wedge s' \oplus \{ ok' \mapsto \mathit{true} \} \in \psi \}
\end{aligned}
$$

The justification of the last step is that

$$\forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \tag{3}$$

is equivalent to

$$\forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \wedge s' \oplus \{ ok' \mapsto \mathit{true} \} \in \psi \tag{4}$$

That (4) implies (3) is simple. To prove the implication in the other direction, we assume that we have an s' such that $(s, s') \in R$. If $s'.ok' = \mathit{true}$, then $s' \oplus \{ok' \mapsto \mathit{true}\} = s'$, which belongs to ψ. If, however, $s'.ok' = \mathit{false}$, then, by $\mathsf{SBH2}$, $(s, s' \oplus \{ok' \mapsto \mathit{true}\}) \in R$. Therefore, by (3), $s' \oplus \{ok' \mapsto \mathit{true}\} \in \psi$ as well. We proceed as follows.

$$
\begin{aligned}
& \{ s \mid \forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \wedge s' \oplus \{ ok' \mapsto \mathit{true} \} \in \psi \} \\
& = \overline{\operatorname{dom}(R \rhd \{ s : \psi \mid s' \oplus \{ ok' \mapsto \mathit{true} \} \in \psi \})} \\
& && \text{[properties of } \rhd\text{, dom, and sets]}
\end{aligned}
$$

$= sb2pt.R.\{\, s : \psi \mid s' \oplus \{ok' \mapsto true\} \in \psi \,\}$ [definition of $sb2pt$]

$\Box$

Theorem 11. *For every set-based relation R that satisfies* SBH3*, $sb2pt.R$ satisfies* PTH3*. Conversely, for every predicate transformer PT that satisfies* PTH3*, $pt2sb.PT$ satisfies* SBH3*.*

Proof. We consider a postcondition $\psi \neq S_{out\alpha PT}$.

$sb2pt.R.\{\, s' : \psi \mid s'.ok' = true \,\}$

$= \overline{\operatorname{dom} R \rhd \{\, s' : \psi \mid s'.ok' = true \,\}}$ [definition of $sb2pt$]

$= \{\, s \mid \forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \wedge s'.ok' = true \,\}$ [properties of $\rhd$, dom, and sets]

$= \{\, s \mid \forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \,\}$ [$\psi \neq S_{out\alpha PT}$ and SBH3]

The details of the last step are as follows. We need to prove that

$$\forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \wedge s'.ok' = true \tag{5}$$

is equivalent to

$$\forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \tag{6}$$

Trivially, (5) implies (6). From (6) and $\psi \neq S_{out\alpha PT}$, we deduce $\exists s' \bullet (s, s') \notin R$. Therefore, by SBH3 and contraposition, $\neg \exists s' \bullet (s, s') \in R \wedge s'.ok' = false$. So, by predicate calculus, $\forall s' \bullet (s, s') \in R \Rightarrow s'.ok' = true$. Consequently, we can conclude (5).

$\{\, s \mid \forall s' \bullet (s, s') \in R \Rightarrow s' \in \psi \,\}$

$= sb2pt.R.\psi$ [definition of $sb2pt$]

For the converse, we consider an $s \in S_{in\alpha PT}$.

$\exists s' \bullet s'.ok' = false \wedge (s, s') \in pt2sb.PT$

$= \exists s' \bullet s'.ok' = false \wedge s \in \overline{PT.\overline{\{s'\}}}$ [definition of $pt2sb$]

$\Rightarrow \exists s' \bullet s \in \overline{PT.\{\, s' \mid s'.ok' = true \,\}}$ [$s'.ok' = false$ and PTH3]

$= s \notin PT.\{\, s' \mid s'.ok' = true \,\}$ [predicate calculus and property of sets]

We now consider a final state s''.

$PT.\overline{\{\, s'' \,\}}$

$= PT.\{\, s' : \overline{s''} \mid s'.ok' = true \,\}$ [PTH3]

$\subseteq PT.\{\, s' \mid s'.ok' = true \,\}$ [monotonicity of PT]

So, from $s \notin PT.\{\, s' \mid s'.ok' = true \,\}$, we conclude that $s \notin PT.\overline{\{\, s'' \,\}}$ either. So, by the definition of $pt2sb$, we have $(s, s'') \in pt2sb.PT$, for all s''.

$\Box$

These healthiness conditions restrict the behaviour of the predicate transformers for postconditions different from $S_{out\alpha PT}$. This particular postcondition, however, is of special interest.

Universally conjunctive predicate transformers can only model terminating programs; we can conclude this from our isomorphisms. In the context of standard predicate transformers wp we can justify this with the observation that universal conjunctivity requires that (2) holds for empty $\{ i \bullet \psi_i \}$. In this case, the intersection $\bigcap\{ i \bullet \psi_i \}$ is the universal set of final states, and the intersection $\bigcap\{ i \bullet wp.\psi_i \}$ is the universal set of initial states. In words, for a 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 $S_{out\alpha PT}$ does not require termination: it accepts any final state s', even those for which $s'.ok' = \mathit{false}$. Similarly, the precondition $S_{in\alpha PT}$ does not even require the program to start.

Therefore, the universal conjunctivity of the predicate transformers corresponding to designs does not imply that only terminating programs can be modelled. Unfortunately, conjunctivity is still an issue: the predicate transformers that are PTH1 and PTH2 healthy are conjunctive. As a consequence, they cannot model angelic nondeterminism.

The relation between predicate transformers that involve ok and ok' and standard predicate transformers is established by the functions $ptok2wp$ and $wp2ptok$ that we present below. In their definitions, we use functions okt and $ok't$, which set to $true$ the ok and ok' components of all states in a given set; dok and dok' remove these components of all states of a set.

Definition 4.

$$ptok2wp.PTOK.\psi \mathrel{\widehat{=}} dok.(PTOK.(ok't.\psi))$$

$$wp2ptok.WP.\psi = \begin{cases} S_{in\alpha}, \text{ if } \psi = S_{out\alpha} \\ okt.(WP.(dok'.\{ s' : \psi \mid s'.ok' = true \})), \text{ if } \psi \neq S_{out\alpha} \end{cases}$$

The function $ptok2wp$ converts one of our predicate transformers $PTOK$ into a standard weakest precondition. This is achieved by extending the alphabet of the states of any postcondition ψ to include ok', and using $PTOK$. In all states of the postcondition with extended alphabet, ok' is associated to **true**, since, in the standard setting, the postconditions implicitly require termination. The alphabet of the states of the resulting precondition includes ok, which is eliminated. No information is lost in the case of healthy $PTOK$ due to PTH1.

The function $wp2ptok$ converts a standard precondition WP to a predicate transformer on ok and ok'. In its definition, the special postcondition $S_{out\alpha}$ is singled out; the corresponding weakest precondition is $S_{in\alpha}$. For other postconditions, we remove the states that require non-termination, reduce the alphabet of the states in the resulting set, and apply WP. The result is a set of states whose alphabet does not include ok; we extend the alphabet and give the value $true$ to ok in all states.

It is not difficult to prove that $ptok2wp$ and $wp2ptok$ establish an isomorphism between the set of conjunctive weakest preconditions and our healthy predicate transformers. We have already seen that designs do not identify non-termination with chaotic behaviour in which any final state is a possibility. This may give the impression that they capture partial correctness, but this is not really the case. Designs allow preconditions to refer to final states, and we have given an example that this leads to programs of uncontrollable behaviour, and not to partial specifications. When we consider H3-healthy designs, we get a model isomorphic to standard weakest preconditions. In [9], different healthiness conditions that lead to a theory of general correctness are proposed.

5 Binary multirelations

The set-based model of binary multirelations is composed of pairs $(\alpha BM, BM)$, where αBM is the alphabet, and BM is a relation between $S_{in\alpha BM}$ and postconditions: elements of $\mathbb{P}\, S_{out\alpha BM}$. 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.

$$\mathsf{BMH} \quad \forall\, s, \psi_1, \psi_2 \mid (s, \psi_1) \in BM \wedge \psi_1 \subseteq \psi_2 \bullet (s, \psi_2) \in BM$$

In [19], there is a proof that a similar model of binary multirelations is isomorphic to a model of monotonic predicate transformers. Here, we consider the isomorphism characterised by the functions below.

Definition 5. $bm2pt.BM.\psi = \{\, s \mid (s, \psi) \in BM \,\}$

$$pt2bm.PT = \{\, (s, \psi) \mid s \in PT.\psi \,\}$$

The function $bm2pt$ converts a binary multirelation to a weakest precondition: $bm2pt.BM$ is guaranteed to establish a postcondition ψ in all initial states s associated to ψ in BM; in these states BM will angelically choose to establish ψ if required. Conversely, the multirelation $pt2bm.PT$ associates an initial state s with all the postconditions that PT is guaranteed to establish from s.

It is not difficult to prove that $bm2pt$ and $pt2bm$ characterise an isomorphism between predicate transformers and binary multirelations; moreover, monotonic predicate transformers correspond to BMH-healthy multirelations.

Theorem 12. *For monotonic P, $pt2bm$ is BMH.*

Proof. Let s, ψ_1, and ψ_2 be such that $(s, \psi_1) \in pt2bm.PT$ and $\psi_1 \subseteq \psi_2$.

$$(s, \psi_1) \in pt2bm.PT$$

$$= s \in PT.\psi_1 \qquad \text{[definition of } pt2bm \text{ and property of sets]}$$

$$\Rightarrow s \in PT.\psi_2 \qquad [\psi_1 \subseteq \psi_2 \Rightarrow PT.\psi_1 \subseteq PT.\psi_2]$$

$(s, \psi_2) \in pt2bm.PT$ [definition of $pt2bm$ and property of sets]

□

Theorem 13. *For healthy BM, $bm2pt.BM$ is monotonic.*

Proof. Let ψ_1 and ψ_2 be such that $\psi_1 \subseteq \psi_2$.

$bm2pt.BM.\psi_1$

$= \{ s \mid (s, \psi_1) \in BM \}$ [definition of $bm2pt$]

$\subseteq \{ s \mid (s, \psi_1) \in BM \}$ [$(s, \psi_1) \in BM \land \psi_1 \subseteq \psi_2 \Rightarrow (s, \psi_2) \in BM$]

$= bm2pt.BM.\psi_2$ [definition of $bm2pt$]

□

What we need is a way of expressing multirelations as alphabetised predicates.

The solution lies in an appropriate choice of alphabet. We propose a view of binary multirelations as a relation between a state on an alphabet $in\alpha$ 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 $out\alpha$. It is simple to establish an isomorphism between this model and binary multirelations: the value of dc' is a postcondition that can be guaranteed by the program. The function $pt2r$ below converts a predicate transformer to a relation with alphabet $in\alpha \cup \{ dc' \}$.

Definition 6. $pt2r.PT = \{ s : S_{in\alpha};\ s' : S_{\{ dc' \}} \mid s \in PT.(s'.dc') \}$

In this context, we can use the functions $pt2r$ and $sb2p$ (Definition 1) to calculate the predicative relational rendering of predicate transformers.

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

Theorem 14. $sb2p.(pt2r.abort) =$ **false**.

Proof.

$sb2p.(pt2r.abort)$

$= sb2p.\{ s, s' \mid s \in abort.(s'.dc') \}$ [definition of $pt2r$]

$= sb2p.\emptyset$ [definition of $abort$]

$= \exists s, s' \bullet (s, s') \in \emptyset \land (\bigwedge x : in\alpha \bullet x = s.x) \land dc' = s'.dc'$ [definition of $sb2p$]

$= false$ [property of sets]

□

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

The healthiness condition PBMH is $P;\ dc \subseteq dc' = P$. This requires that, if, after executing P, we execute a program that enlarges dc', then the result could have been obtained by P itself. This means that P characterises dc' not by defining a particular value, but the smallest set of elements it should include.

An interesting example is a design $P \vdash Q$. Its definition as a predicate transformer is as follows.

$$(P \vdash Q).\psi = c2sb.P \cap (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup \psi) \dagger in\alpha$$

The function $c2sb$ transforms predicates P on undashed variables into sets of states (see Definition 3) The function $c'2sb$ is similar, but handles predicates on dashed variables.

$$c'2sb.C = \{\, s' \mid C[s'/in\alpha] \,\}$$

The universal quantification is defined for the set-based representation of predicates as follows.

$$s' \in (\overline{\forall} x : out\alpha \bullet SS) \Leftrightarrow \forall\, v \bullet s' \oplus \{\, x : out\alpha \bullet x \mapsto v \,\} \in SS$$

Finally, the $\dagger$ operator extends the alphabet of the states in a set. In the definition of design, the alphabet of the states in the universal quantification is empty. There is only one state on the empty alphabet: the empty state. If the quantification holds, the set it defines contains the empty state; otherwise it defines the empty set. Extending the alphabet is needed for the conjunction with $c2sb.P$.

$$s_1 \in P \dagger A \Leftrightarrow \exists\, s_2 \bullet\!: P \bullet A \lhd s_1 = A \lhd s_2$$

The state $A \lhd s$ is obtained by removing all the components in A from s.

The following theorem establishes how designs can be characterised in the UTP using binary multirelations.

Theorem 15. $sb2p.(pt2r.(P \vdash Q)) = P \wedge \{\, s' \mid Q \,\} \subseteq dc'$

Proof. We use the notation $(\theta x : A \bullet x \mapsto v)$ to denote the state with components x from an alphabet A, each of which has the value given by v.

$$sb2p.(pt2r.(P \vdash Q))$$

$$= sb2p.\{\, s, s' \mid s \in (P \vdash Q).(s'.dc') \,\} \qquad \text{[definition of } pt2r\text{]}$$

$$= sb2p.\{\, s, s' \mid s \in c2sb.P \cap (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup s'.dc') \dagger in\alpha$$

$$\text{[definition of } P \vdash Q\text{]}$$

$$\begin{aligned} = \exists\, s, s' \bullet s \in c2sb.P \cap (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup s'.dc') \dagger in\alpha \wedge \\ (\textstyle\bigwedge x : in\alpha R \bullet x = s.x) \wedge s'.dc' = dc' \end{aligned}$$

$$\text{[definition of } sb2p\text{]}$$

$$= \exists\, s \bullet s \in c2sb.P \cap (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup dc') \dagger in\alpha \land$$
$$(\textstyle\bigwedge x : in\alpha R \bullet x = s.x)$$

[equality of records and one-point rule]

$$= (\theta x : in\alpha \bullet x \mapsto x) \in c2sb.P \cap (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup dc') \dagger in\alpha$$

[equality of records and one-point rule]

$$= (\theta x : in\alpha \bullet x \mapsto x) \in c2sb.P \land$$
$$(\theta x : in\alpha \bullet x \mapsto x) \in (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup dc') \dagger in\alpha$$ [property of sets]

$$= (\theta x : in\alpha \bullet x \mapsto x) \in \{s \mid P[s/in\alpha]\} \land$$
$$(\theta x : in\alpha \bullet x \mapsto x) \in (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup dc') \dagger in\alpha$$ [definition of $c2sb$]

$$= P \land (\theta x : in\alpha \bullet x \mapsto x) \in (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup dc') \dagger in\alpha$$

[property of sets and substitution]

$$= P \land \exists\, s_2 : (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup dc') \bullet$$
$$in\alpha \lhd (\theta x : in\alpha \bullet x \mapsto x) = in\alpha \lhd s_2$$ [definition of $\dagger$]

$$= P \land \exists\, s_2 \bullet s_2 \in (\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup dc')$$

[$\overline{\forall} x : out\alpha \bullet \overline{c'2sb.Q} \cup dc'$ is either $\emptyset$ or contains the empty state]

$$= P \land \exists\, s_2 \bullet \forall\, v \bullet s_2 \oplus \{\, x : out\alpha \bullet x \mapsto v \,\} \in \overline{c'2sb.Q} \cup dc'$$ [definition of $\overline{\forall}$]

$$= P \land \exists\, s_2 \bullet \forall\, v \bullet (\theta\, x : out\alpha \bullet x \mapsto v\,) \in \overline{c'2sb.Q} \cup dc'$$

[s_2 has no components]

$$= P \land$$
$$\exists\, s_2 \bullet \forall\, v \bullet (\theta\, x : out\alpha \bullet x \mapsto v\,) \notin c'2sb.Q \lor (\theta\, x : out\alpha \bullet x \mapsto v\,) \in dc'$$

[property of sets]

$$= P \land \exists\, s_2 \bullet \forall\, v \bullet (\theta\, x : out\alpha \bullet x \mapsto v\,) \notin \{\, s' \mid Q[s'/out\alpha] \,\} \lor$$
$$(\theta\, x : out\alpha \bullet x \mapsto v\,) \in dc'$$

[definition of $c'2sb$]

$$= P \land \exists\, s_2 \bullet \forall\, v \bullet \neg\, Q[v/out\alpha] \lor (\theta\, x : out\alpha \bullet x \mapsto v\,) \in dc'$$

[property of sets and substitution]

$$= P \land \exists\, s_2 \bullet \forall\, x : out\alpha \bullet Q \Rightarrow (\theta\, x : out\alpha \bullet x \mapsto x\,) \in dc'$$

[predicate calculus]

$$= P \land\ s' \mid Q\ \subseteq dc'$$ [property of sets]

□

For $P \vdash Q$ to terminate, P has to hold. In this case, Q characterises the demonic choices available; healthiness requires that dc' is allowed to be any superset of the set of states that satisfy Q. The proof that healthy binary multirelations correspond to PBMH-healthy predicates is not difficult.

Angelic choice $P \sqcup Q$ is characterised by disjunction.

Theorem 16. $sb2p.(pt2r.(P \sqcup Q)) = sb2p.(pt2r.P) \vee sb2p.(prt2.Q)$

Proof.

$$
\begin{aligned}
& sb2p.(pt2r.(P \sqcup Q)) \\
&= sb2p.\{\, s, s' \mid s \in (P \sqcup Q).(s'.dc') \,\} \\
&= sb2p.\{\, s, s' \mid s \in P.(s'.dc') \cup Q.(s'.dc') \,\} \\
&= \exists\, s, s' \bullet s \in P.(s'.dc') \cup Q.(s'.dc') \wedge (\textstyle\bigwedge x : in\alpha R \bullet x = s.x) \wedge s'.dc' = dc' \\
&\hfill [\text{definition of } sb2p] \\
&= (\exists\, s, s' \bullet s \in P.(s'.dc') \wedge (\textstyle\bigwedge x : in\alpha R \bullet x = s.x) \wedge s'.dc' = dc') \vee \\
&\quad (\exists\, s, s' \bullet s \in Q.(s'.dc') \wedge (\textstyle\bigwedge x : in\alpha R \bullet x = s.x) \wedge s'.dc' = dc') \\
&\hfill [\text{property of sets and predicate calculus}] \\
&= sb2p.(pt2r.P) \vee sb2p.(pt2r.Q) \qquad [\text{definitions of } pt2r \text{ and } sb2p]
\end{aligned}
$$

□

The program $P \sqcup Q$ gives all the guarantees that can be provided by choosing P, together with those that arise from the possibility of choosing Q. Demonic choice is captured by conjunction; a postcondition is guaranteed by $P \sqcap Q$ only if both P and Q can guarantee it, so that the arbitrary choice is not a problem.

Sequential composition cannot correspond to relational composition because the relations are not homogeneous. The definition of $P;\ Q$ uses the operator * to lift Q to a predicate on dc and dc'. It is inspired on a similar UTP operator used in the treatment of logic programming, and is defined as follows.

$$
\begin{aligned}
Q^* \;\widehat{=}\;\; & \mu\, X \bullet dc' = \emptyset \lhd dc = \emptyset \rhd \mathbf{var}\ s \bullet s' \in dc; \\
& \qquad\qquad\qquad\qquad (v := s.v;\ Q) \sqcup (dc := dc \setminus \{\, s \,\};\ X) \\
& \qquad\qquad\qquad \mathbf{end}
\end{aligned}
$$

The definition of $P;\ Q$ is $P;\ Q^*$. After the execution of P, Q^* recursively selects a state in dc' and executes Q. The program $P \lhd c \rhd Q$ 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 complicated than those in the original UTP model. The refinement relation is not implication anymore either. 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.

6 Conclusions

The central objective of Hoare and He's unifying theories of programming [13] is to formalise different programming paradigms within a common semantic framework, so that they may be directly compared. In this way new compound programming 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 theoretical and practical programming paradigm: it plays an important rôle in the theory of refinement calculi; and it is used as an abstraction in search-based and constraint-oriented programming, hiding details of how particular strategies are implemented. The main contributions of this report are the formal demonstration that angelic nondeterminism cannot be directly expressed within the UTP, as presented in [13], and in the predicative account of binary multirelations that allows the unification of angelic nondeterminism.

In Section 2, we describe Hoare and He's semantic framework, the general theory of alphabetised relations, presented in a predicative style. We give a brief introduction to the subtheory of designs, where it is possible to observe both the start of a program and also its termination, and we describe healthiness conditions for that subtheory. Designs enable us to reason about total correctness, and in Section 3, we give a set-based model for the relational calculus that brings this fact sharply into focus.

We show that the theory of general relations is isomorphic to our set-based model, which is clearly a model of terminating programs. Continuing in this vein, we build a set-based model that is isomorphic to the subtheory of designs. In doing so, we prove a modest but novel result that shows that one of Hoare and He's healthiness conditions is subsumed by another.

It is known that an isomorphism exists between certain predicate transformers and relations, and in Section 4, we turn our attention to predicate transformers, which we model as functions on sets of states. We show that there is an isomorphism between our set-based relations and the subtheory of universally conjunctive predicate transformers. The significance of this is the connection that it establishes with a result due to Back and von Wright: the conjunctive predicate transformer setting cannot capture angelic nondeterminism.

The subtheory of designs is interesting: although designs are merely relations, the subtheory has a rich structure imposed by the special interpretation of the observations ok and ok' and their associated healthiness conditions. This structure allows us to describe non-termination; but what else besides? Whenever we introduce new observations, we must investigate the consequences. The point here is this: can designs describe angelic nondeterminism? Our investigations show that there is an isomorphism between designs and conjunctive predicate transformers, so the answer is negative.

A relational model that can capture both angelic and demonic nondeterminism is presented in [19]. 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 [22], and prove refinement laws.

The work in [15] presents a functional semantics for a tactic language which includes angelic nondeterminism. In that semantics, the semantics of angelic choice is a list that contains all the options available to the angel. That language, however, does not include demonic nondeterminism. In [16], the set-based model of binary relations is used to support angelic and demonic nondeterminism in a calculus for functional programs.

Acknowledgements

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

References

1. R. J. R. Back 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*, volume 375 of *Lecture Notes in Computer Science*, pages 139 – 156, Groningen, The Netherlands, 1989. Springer-Verlag.
2. R. J. R. Back and J. Wright. Duality in Specification Languages: A Lattice-theoretical Approach. *Acta Informatica*, 27(7):583 – 625, 1990.
3. R. J. R. Back and J. Wright. Combining angels, demons and miracles in program specifications. *Theoretical Computer Science*, 100:365 – 383, 1992.
4. R. J. R. Back and J. Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag, 1998.
5. A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. A Refinement Strategy for *Circus*. *Formal Aspects of Computing*, 15(2 – 3):146 – 181, 2003.
6. A. L. C. Cavalcanti and J. C. P. Woodcock. A Weakest Precondition Semantics for Z. *The Computer Journal*, 41(1):1 – 15, 1998.
7. A. L. C. Cavalcanti and J. C. P. Woodcock. ZRC—A Refinement Calculus for Z. *Formal Aspects of Computing*, 10(3):267—289, 1999.
8. E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
9. S. Dunne. 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, July 2001.
10. P. H. B. Gardiner and C. C. Morgan. Data Refinement of Predicate Transformers. *Theoretical Computer Science*, 87:143 – 162, 1991.
11. W. H. Hesselink. *Programs, Recursion and Unbounded Choice – Predicate Transformation Semantics and Transformation Rules.* Cambridge Tracts in Theoretical Computer Science 27. Cambridge University Press, 1992.
12. C. A. R. Hoare and Jifeng He. The Weakest Prespecification. Technical Monograph TM-PRG-44, Oxford University Computing Laboratory, Oxford – UK, 1985.
13. C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall, 1998.

14. R. Jagadeesan, V. Shanbhogue, and V. Saraswat. Angelic non-determinism in concurrent constraint programming. Technical report, Xerox Park, January 1991.
15. A. P. Martin, P. H. B. Gardiner, and J. C. P. Woodcock. A Tactical Calculus. *Formal Aspects of Computing*, 8(4):479–489, 1996.
16. C. E. Martin, S. A. Curtis, and I. Rewitzky. Modelling Nondeterminism. In *Mathematics of Program Construction*, Lecture Notes in Computer Science, 2004.
17. C. C. Morgan. *Programming from Specifications*. Prentice-Hall, 2nd edition, 1994.
18. C. C. Morgan and P. H. B. Gardiner. Data Refinement by Calculation. *Acta Informatica*, 27(6):481—503, 1990.
19. I. Rewtizky. Binary Multirelations. In H. Swart, E. Orlowska, G. Schmidt, and M. Roubens, editors, *Theory and Application of Relational Structures as Knowledge Instruments*, volume 2929 of *Lecture Notes in Computer Science*, pages 256 – 271, 2003.
20. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall Series in Computer Science. Prentice-Hall, 1998.
21. J. C. P. Woodcock and A. L. C. Cavalcanti. The steam boiler in a unified theory of Z and CSP. In *8th Asia-Pacific Software Engineering Conference (APSEC 2001)*. IEEE Press, 2001.
22. J. C. P. Woodcock 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*, volume 2272 of *Lecture Notes in Computer Science*, pages 184—203. Springer-Verlag, 2002.
23. J. C. P. Woodcock and A. L. C. Cavalcanti. A Tutorial Introduction to Designs in Unifying Theories of Programming. In *IFM 2004: Integrated Formal Methods*, volume 2999 of *Lecture Notes in Computer Science*, pages 40 – 66. Springer-Verlag, 2004. Invited tutorial.
24. J. C. P. Woodcock and J. Davies. *Using Z—Specification, Refinement, and Proof.* Prentice-Hall, 1996.