Designs with angelic nondeterminism

Pedro Ribeiro
Department of Computer Science
University of York
York, UK
E-mail: pfr500@york.ac.uk

Ana Cavalcanti
Department of Computer Science
University of York
York, UK
E-mail: ana.cavalcanti@york.ac.uk

***Abstract*—Hoare and He's Unifying Theories of Programming (UTP) are a predicative relational framework for the definition and combination of refinement languages for a variety of programming paradigms. Previous work has defined a theory for angelic nondeterminism in the UTP; this is basically an encoding of binary multirelations in a predicative model. In the UTP a theory of designs (pre and postcondition pairs) provides, not only a model of terminating programs, but also a stepping stone to define a theory for state-rich reactive processes. In this paper, we cast the angelic nondeterminism theory of the UTP as a theory of designs with the long-term objective of providing a model for well established refinement process algebras like Communicating Sequential Processes (CSP) and *Circus*.**

I. Introduction

In [1] Hoare and He introduce the UTP, a predicative framework of alphabetized relations suitable for characterising and reasoning about programs based on the principle of observation. Relations are characterised by their alphabet and a predicate whose free variables determine the possible observations of a particular mechanism. These variables can be either program variables or auxiliary variables that record additional information, including for instance, time.

A collection of UTP theories are presented in [1] that target multiple aspects of different programming paradigms, such as functionality, concurrency, logic programming and high-order programming. In addition, other UTP theories have been proposed that can handle angelic-nondeterminism [2], object-orientation [3], pointers [4], time [5]–[7] and others. The central aspect of the UTP is that theories can be linked and their results reused. This promotes unification while allowing different aspects of programs to be considered in isolation.

Total correctness of sequential programs is characterised in the UTP through the theory of designs that captures the traditional pre and postcondition specification pairs. Termination is modelled using auxiliary boolean variables.

While the relationship between the initial and final value of program variables is sufficient to provide semantics for sequential programs, in the case of reactive processes intermediate information also needs to be recorded. This is handled in the UTP by the theory of reactive processes, whose alphabet includes extra observational variables for this purpose. The combination of the theory of designs and the theory of reactive processes supports the characterisation of CSP [8]. Every predicate of the theory of CSP can be specified as a reactive design [1], [9] with pre and postcondition pairs.

Angelic nondeterminism is a useful specification construct that allows for a high degree of abstraction to be achieved in specifications. It has traditionally been studied within the refinement calculus [10]–[12] through the monotonic predicate transformers. There it is defined in terms of weakest precondition semantics, and it is precisely the dual of demonic nondeterminism. Unfortunately it is not possible to characterise both forms of nondeterminism directly in a relational setting [2], [13] such as that of the UTP, instead multirelational models can be used [2], [14].

In [14] Rewitzky presents the foundational work on multirelations to model both forms of nondeterminism within a single relational model. Multirelations relate input states to sets of states. The most important model is that of up-closed multirelations that has a lattice-theoretic structure. Refinement notions are further elaborated in [15].

Cavalcanti et al. [2] propose a UTP theory based on multirelations that can encode angelic nondeterminism. It is focused on correctness of sequential programs and not applicable to reactive programs. The theory in [2] is, first of all, cast in the general set of relations, which can only treat partial correctness. The encoding of binary multirelations, on the other hand, can cater for termination and designs are not really considered as a separate theory.

The contribution of this work is a new UTP theory of designs that can express both angelic and demonic nondeterminism. Our new theory of designs is suitable for combination with the theory of reactive designs, and, therefore, for definition of a process algebra with angelic nondeterminism. We observe that the theory of designs encompasses programs whose precondition depends not only on observations of the initial values, but also on the final or later values of the variables. This is required to establish the link with the theory of CSP. Our theory, therefore, addresses the challenge of catering for such preconditions in the presence of angelic nondeterminism. In this context, for example, the definition of sequential composition is not immediately obvious.

The structure of this paper is as follows. Section II briefly introduces the UTP, the theory of designs and contextualizes this work by introducing the encoding of [2]. In Section III the new theory is introduced by defining its alphabet and its healthiness conditions. Section IV discusses the operators, their basic properties and provides some intuition with the aid of examples. In Section V we show how our theory relates to that of [2]. Finally in Section VI we present our conclusions.

II. UTP

In the UTP [1], a theory is defined by three essential elements: an alphabet, which consists of a set of variables corresponding to observations made of programs; a set of healthiness conditions, usually specified as idempotent monotonic functions whose fixed points are the valid predicates of the theory; and a set of operators.

The alphabet is split into two subsets, where undashed and dashed variables characterise initial and final states, respectively. The input alphabet corresponds to the set of undashed variables while the output alphabet corresponds to the set of dashed variables. For a relation where both alphabets are exactly the same, except for the fact that they are dashed or undashed, it is said to be homogeneous. For theories where relations are homogenenous the sequential composition operator is defined as relational composition.

A. UTP designs

As introduced by Hoare and He [1], the theory of designs is the definitive treatment of total correctness for sequential programs. Its alphabet consists of the program variables and two auxiliary Boolean variables ok and ok' that record when a program starts, and when it finishes, respectively. The definition of a design is reproduced below.

Definition 1:

$$(P_0 \vdash P_1) \mathrel{\widehat{=}} (ok \wedge P_0) \Rightarrow (P_1 \wedge ok')$$

The relation P_0 corresponds to the precondition, while P_1 corresponds to the postcondition. A design P can be expressed in this form if and only if it satisfies the healthiness conditions **H1** and **H2** [1] (whose composition we call simply **H**) as reproduced below, where $P[E/b]$ is the predicate obtained by substitution of every free variable b in P with expression E.

Definition 2:

$$\mathbf{H}(P) = (ok \wedge \neg\, P[false/ok']) \Rightarrow (P[true/ok'] \wedge ok')$$

This is more concisely written using the following shorthand notation $P^a = P[a/ok']$, with $t = true$ and $f = false$, as introduced by Woodcock and Cavalcanti [16].

$$\mathbf{H}(P) = (\neg\, P^f \vdash P^t)$$

We observe that ok is not free in either P^f or P^t [1] and that given a design P its precondition is given by $\neg\, P^f$ and its postcondition by P^t. In addition we also note that none of the proofs [1], [16] for idempotency and monotonicity regarding **H1**, **H2** and **H** rely on relations being homogeneous.

When modelling sequential programs, the precondition P of a design is in fact not a relation, but rather a condition that only refers to undashed variables. Designs exhibiting this characteristic are fixed points of the healthiness condition **H3** whose definition [1] we reproduce below.

Definition 3:

$$\mathbf{H3}(P) = P\,;(true \vdash x' = x)$$

The requirement imposed by **H3** is that the skip of the theory, defined as $(true \vdash x' = x)$, is a right-unit for sequential composition [1]. In order to see the consequences of a non-**H3**-design consider the following example.

Example 1:

$$\begin{aligned}&(x' \neq 2 \vdash x' = 1)\\ &= (ok \wedge x' \neq 2) \Rightarrow (x' = 1 \wedge ok')\\ &= ok \Rightarrow ((x' = 1 \wedge ok') \vee x' = 2)\end{aligned}$$

This is a design that once started, that is ok is *true*, can either establish $x' = 1$ and terminate with ok' being *true*, or alternatively can establish $x' = 2$ without necessarily terminating. This is a behaviour that would not normally be expected in a theory for sequential programs. However, reactive programs can be expressed as the image of non-**H3** designs through the function **R** that characterises the theory of reactive programs [17].

B. Designs with angelic nondeterminism

In [2] a non-homogeneous theory of designs is presented that can encode both notions of nondeterminism. Its alphabet consists of the undashed program variables, and of the sole dashed variable, ac'. This dashed variable represents the set of final states available for angelic choice, while the choices of the value of ac' encode the demonic choices. A state is a record whose components represent program variables. For instance, for a program whose only variable is x, then ac' must contain a record component of name x', such that its value is one of the possible final values of x'. Despite not including the variables ok and ok', the model of [2] captures termination.

The only healthiness condition of the theory in [2] is defined by **PBMH**, whose requirement is that the value of ac' is upward closed. This is defined using sequential composition [2].

Definition 4:

$$\mathbf{PBMH}(P) \mathrel{\widehat{=}} P\,;ac \subseteq ac'$$

If it were possible for P to establish some value of ac' then it must also be the case that any superset could have been obtained. This is the same requirement observed in the theory of binary multirelations [14].

In fact, Cavalcanti et al. [2] establish that their theory is isomorphic to both the universally monotonic predicate transformers and binary multirelations. The linking function defined there is used to calculate the definition of the operators from the predicate transformers model. Since that model observes **H3**, then we can ascertain that as a result the isomorphic models can only express **H3**-designs. It is precisely this restriction that we need to avoid.

III. Designs

The new theory that we propose is a theory of designs with non-homogeneous relations. The alphabet we consider contains both ok and ok' like in the original theory of designs. In addition it has two variables s and ac' as shown in the following definition.

Definition 5 (Alphabet):

$$s : State$$
$$ac' : \mathbb{P}\, State$$
$$ok, ok' : \{true, false\}$$

The variable s encapsulates the initial values of program variables as record components of s. The set of final states ac' is similar to that of [2] except that we only consider undashed variables in each state. This deliberate choice bears no consequences, other than simplifying reasoning. In fact, our ac' can be related to and from the original ac' of [2] by either dashing or undashing the variables in all states in either set.

A. Healthiness conditions

Since the theory we propose is a theory of designs, predicates need to satisfy **H**. In addition, since we seek to define a theory of designs that uses ok and ok', a consistent notion of termination needs to be established with respect to ac'. This is addressed by the healthiness condition **A0**.

Furthermore, and similarly to the theory of [2], there is a requirement on ac' to be upward closed. However, because designs are not necessarily **H3**-healthy, this requirement needs to be extended to potentially non-terminating designs. This is the concern of the healthiness condition **A1**.

Finally the new theory is fully characterised by the functional composition of **A0** and **A1** as specified by the function **A**. In what follows we define each of the functions and present their functional composition **A** and its properties.

B. A0: Termination

The notion of termination embodied in our theory is related to that of [2]. In that model, termination is always guaranteed as long as ac' is not empty. In fact, $ac' \neq \emptyset$ closely matches the design *Choice*, where any final outcome is allowed as long as it terminates. In contrast, abortion is characterised in [2] by *true*, as in that case ac' is allowed every possible value.

In our theory, once termination is guaranteed, that is ok' holds, then ac' cannot be empty. This constraint is imposed on a design P by our new healthiness condition **A0**.

*Definition 6 (**A0**):*

$$\mathbf{A0}(P) \mathrel{\widehat{=}} (\neg\, P^f \vdash P^t \wedge ac' \neq \emptyset)$$

This function is idempotent and monotonic with respect to the refinement ordering. In addition it distributes over both conjunction and disjunction. As a result it is also closed with respect to conjunction and disjunction. Proof of this and all other results discussed in this paper can be found in [18].

C. A1: Upward closure

The notion of upward closure needs to be revisited in light of the possibility for non-termination. The requirement upon our theory is expressed by **A1** as defined below.

*Definition 7 (**A1**):*

$$\mathbf{A1}(P_0 \vdash P_1) \mathrel{\widehat{=}} (\neg\, \mathbf{PBMH}(\neg\, P_0) \vdash \mathbf{PBMH}(P_1))$$

The upward closure of ac' in the postcondition, P_1, is enforced exactly as in the theory of [2]. The difference here is how the precondition is handled. In this case we ensure that it is the negation of the precondition that must be upward closed. This is because it is actually the negation of the precondition that establishes the value of ac' when the design does not require termination. We can rewrite **A1** as follows.

Lemma 1:

$$\mathbf{A1}(P_0 \vdash P_1) = ok \Rightarrow \left(\begin{array}{l} ((P_1 \,;\, ac \subseteq ac') \wedge ok') \\ \vee \\ (\neg\, P_0 \,;\, ac \subseteq ac') \end{array} \right)$$

Proof:

$$\begin{array}{ll} & \mathbf{A1}(P_0 \vdash P_1) \qquad \{\text{Definition of } \mathbf{A1}\} \\ = & (\neg\, (\neg\, P_0 \,;\, ac \subseteq ac') \vdash P_1 \,;\, ac \subseteq ac') \\ & \qquad \{\text{Definition of designs}\} \\ = & (ok \wedge \neg\, (\neg\, P_0 \,;\, ac \subseteq ac')) \Rightarrow ((P_1 \,;\, ac \subseteq ac') \wedge ok') \\ & \qquad \{\text{Predicate calculus}\} \\ = & ok \Rightarrow (((P_1 \,;\, ac \subseteq ac') \wedge ok') \vee (\neg\, P_0 \,;\, ac \subseteq ac')) \end{array}$$

■

In both the terminating or potentially non-terminating case, ac' is required to be upward closed.

The function **A1** is idempotent and monotonic with respect to the refinement ordering. It is closed with respect to conjunction and disjunction, provided both operands are **A1**-healthy. Furthermore it also distributes through disjunction.

D. A

The designs of the theory are characterised by the functional composition of **A1** followed by **A0**.

*Definition 8 (**A**):*

$$\mathbf{A}(P) \mathrel{\widehat{=}} \mathbf{A0} \circ \mathbf{A1}(P)$$

The reason for this particular order is that the functions do not always necessarily commute. In order to see the reason consider the following counter-example.

Counter-example 1:

$$\begin{array}{ll} & \mathbf{A0} \circ \mathbf{A1}(true \vdash ac' = \emptyset) \qquad \{\text{Definition of } \mathbf{A1}\} \\ = & \mathbf{A0}(\neg\, (false \,;\, ac \subseteq ac') \vdash ac' = \emptyset \,;\, ac \subseteq ac') \\ & \qquad \{\text{Definition of sequential composition}\} \\ = & \mathbf{A0} \left(\begin{array}{l} \neg\, (false \wedge \exists\, ac_0 \bullet ac_0 \subseteq ac') \\ \vdash \\ \exists\, ac_0 \bullet ac_0 = \emptyset \wedge ac_0 \subseteq ac' \end{array} \right) \\ & \qquad \{\text{One-point rule and predicate calculus}\} \\ = & \mathbf{A0}(true \vdash true) \qquad \{\text{Definition of } \mathbf{A0}\} \\ = & \mathbf{A0}(true \vdash ac' \neq \emptyset) \end{array}$$

$$\begin{array}{ll} & \mathbf{A1} \circ \mathbf{A0}(true \vdash ac' = \emptyset) \qquad \{\text{Definition of } \mathbf{A0}\} \\ = & \mathbf{A1}(true \vdash ac' = \emptyset \wedge ac' \neq \emptyset) \qquad \{\text{Predicate calculus}\} \\ = & \mathbf{A1}(true \vdash false) \qquad \{\text{Definition of } \mathbf{A1}\} \\ = & (\neg\, (false \,;\, ac \subseteq ac') \vdash false \,;\, ac \subseteq ac') \\ & \qquad \{\text{Definition of sequential composition}\} \\ = & (true \vdash false) \end{array}$$

In this example we consider the application of the healthiness conditions to an unhealthy design whose postcondition requires non-termination by requiring $ac' = \emptyset$. In the first case **A1** changes the precondition into true, followed by the application of **A0**. While in the second case, the application of **A0** makes the postcondition false, a predicate that satisfies **PBMH**.

If instead we consider healthy predicates, then **A0** and **A1** commute. The following Lemma 2 establishes this.

Lemma 2: Provided P^t satisfies **PBMH**.

$$\mathbf{A0} \circ \mathbf{A1}(P) = \mathbf{A1} \circ \mathbf{A0}(P)$$

A proof of this result (and a few others to follow) can be found in the Appendix. The only requirement is for the postcondition of P to satisfy **PBMH**. This requirement can be met as long as we apply **A1** first. Since **A0** and **A1** are idempotent, it follows from the commutativity of Lemma 2 that so is **A** [1]. Furthermore, monotonicity follows from that of **A0** and **A1**.

The healthiness condition of our theory is $\mathbf{H} \circ \mathbf{A}$. Since **H** and **A** commute, and **H** and **A** are idempotent, so is $\mathbf{H} \circ \mathbf{A}$. Likewise, monotonicity also follows from that of **H** and **A**.

Furthermore, as **A** is idempotent and monotonic, a result in [1] establishes that such a function also yields a complete lattice. Therefore the theory we propose is also a complete lattice under the refinement ordering of the UTP.

IV. Operators

In this section the operators of the theory are defined. First we define the assignment operator followed by the most important operator sequential composition. Since the theory considers non-homogeneous relations this is also the most challenging operator. This is followed by the definition of demonic and angelic choice operators.

A. Assignment

Similarly to the definition in [2], the assignment operator is defined as follows.

Definition 9 (Assignment):

$$(x :=_{\mathcal{D}ac} e) \mathrel{\widehat{=}} (true \vdash s \oplus (x \mapsto e) \in ac')$$

Its definition is specified by a design whose precondition is true, and whose postcondition establishes that every set of final states ac' available for demonic choice has a component where x is assigned the value of expression e. Every such state is the result of overriding the initial state s on the value of x, thus leaving every other program variable unchanged.

B. Sequential composition

The most challenging aspect of this theory is its reliance on non-homogeneous relations. As explained earlier, the consequence is that sequential composition cannot be defined as relational composition. The definition that we introduce below is layered upon that of the sequential composition in [2].

Definition 10 ($;_{\mathcal{D}\mathbf{ac}}$-sequence):

$$P \;;_{\mathcal{D}\mathbf{ac}}\; Q \mathrel{\widehat{=}} \exists\, ok_0 \bullet P[ok_0/ok'] \;;_{\mathcal{A}}\; Q[ok_0/ok]$$

This definition resembles relational composition with the notable difference that instead of conjunction another operator is used ($;_{\mathcal{A}}$) that handles the non-homogeneous alphabet of the relations. This operator actually closely corresponds to the sequential composition operator introduced in [2]. The difference lies in how we treat the initial state with the variable s instead of individual program variables.

Definition 11 ($;_{\mathcal{A}}$-sequence):

$$P \;;_{\mathcal{A}}\; Q \mathrel{\widehat{=}} P[\{z : State \mid Q[z/s]\}/ac']$$

This sequential composition can be understood as follows: a final state of $P \;;_{\mathcal{A}}\; Q$ is a final state of Q that can be reached from a set of input states z of Q that is available to P as a set ac' of angelic choices.

A more intuitive interpretation can be given by considering the operator $;_{\mathcal{A}}$ as back propagating the information concerning the valid final states, thus resembling a backtracking operation. Consider the following example from [2].

Example 2:

$$(s \oplus (x \mapsto 1)) \in ac' \;;_{\mathcal{A}} \left(\begin{array}{l} (s \oplus (x \mapsto s.x + 1)) \in ac' \\ \wedge \\ (s \oplus (x \mapsto s.x + 2)) \in ac' \end{array} \right)$$

{Definition of $;_{\mathcal{A}}$ and substitution}

$$= (s \oplus (x \mapsto 1)) \in \left\{ z \;\middle|\; \begin{array}{l} ((s \oplus (x \mapsto s.x + 1)) \in ac')[z/s] \\ \wedge \\ ((s \oplus (x \mapsto s.x + 2)) \in ac')[z/s] \end{array} \right\}$$

{Substitution}

$$= (s \oplus (x \mapsto 1)) \in \left\{ z \;\middle|\; \begin{array}{l} (z \oplus (x \mapsto z.x + 1)) \in ac' \\ \wedge \\ (z \oplus (x \mapsto z.x + 2)) \in ac' \end{array} \right\}$$

{Property of sets}

$$= \left(\begin{array}{l} (s \oplus (x \mapsto 1) \oplus (x \mapsto (s \oplus (x \mapsto 1)).x + 1)) \in ac' \\ \wedge \\ (s \oplus (x \mapsto 1) \oplus (x \mapsto (s \oplus (x \mapsto 1)).x + 2)) \in ac' \end{array} \right)$$

{Record component}

$$= \left(\begin{array}{l} (s \oplus (x \mapsto 1) \oplus (x \mapsto 2)) \in ac' \\ \wedge \\ (s \oplus (x \mapsto 1) \oplus (x \mapsto 3)) \in ac' \end{array} \right) \qquad \{\text{Property of } \oplus\}$$

$$= (s \oplus (x \mapsto 2)) \in ac' \wedge (s \oplus (x \mapsto 3)) \in ac'$$

In this example we consider the sequential composition of a predicate that assigns 1 to x, followed by the conjunction of two predicates: one that increments the initial value of x by one, and the other by two. We observe that in [2] conjunction corresponds to angelic choice. If we take that interpretation, then the sequential composition yields two choices for assigning a value to x in ac' available to the angel.

Based on properties of the operator $;_{\mathcal{A}}$ it is possible to characterise the sequential composition of **A**-healthy designs as established by the following Theorem 1.

Theorem 1: Provided $(P_0 \vdash P_1)$ is **A**-healthy.

$$(P_0 \vdash P_1) \;;_{\mathcal{D}\mathbf{ac}}\; (Q_0 \vdash Q_1)$$
$$=$$
$$(\neg\, (\neg\, P_0 \;;_{\mathcal{A}}\; true) \wedge \neg\, (P_1 \;;_{\mathcal{A}}\; \neg\, Q_0) \vdash (P_1 \;;_{\mathcal{A}}\; Q_1))$$

The result obtained is similar to that of sequential composition for the original theory of designs [1], [16], except for the use of the operator $;_{\mathcal{A}}$ instead of relational composition [1].

Similarly to the original theory of designs, we identify the *Skip* of the theory whose definition we present below.

Definition 12 (Skip):

$$\mathit{II}_{\mathcal{D}_{\mathbf{ac}}} \mathrel{\widehat{=}} (\mathit{true} \vdash s \in ac')$$

This is a design that always terminates and upon termination establishes that the input state s is in all sets of angelic choices ac'. The behaviour of $\mathit{II}_{\mathcal{D}_{\mathbf{ac}}}$ is to maintain the current state. As expected $\mathit{II}_{\mathcal{D}_{\mathbf{ac}}}$ is **A**-healthy and is the left-unit for sequential composition.

Law 1 (Skip-$;_{\mathcal{D}_{\mathbf{ac}}}$): Provided P is a design.

$$\mathit{II}_{\mathcal{D}_{\mathbf{ac}}} ;_{\mathcal{D}_{\mathbf{ac}}} P = P$$

This law can be proved from the definition of $\mathit{II}_{\mathcal{D}_{\mathbf{ac}}}$ and by application of Theorem 1. It confirms the suitability of $\mathit{II}_{\mathcal{D}_{\mathbf{ac}}}$ as the *Skip* of the theory.

Finally, it is also possible to show that $;_{\mathcal{D}_{\mathbf{ac}}}$ behaves as expected with respect to the extreme points of the lattice.

Law 2 ($\bot_{\mathcal{D}}-$ $;_{\mathcal{D}_{\mathbf{ac}}}$):

$$\bot_{\mathcal{D}} ;_{\mathcal{D}_{\mathbf{ac}}} P = \bot_{\mathcal{D}}$$

Law 3 ($\top_{\mathcal{D}}-$ $;_{\mathcal{D}_{\mathbf{ac}}}$):

$$\top_{\mathcal{D}} ;_{\mathcal{D}_{\mathbf{ac}}} P = \top_{\mathcal{D}}$$

This follows directly from the definition of $;_{\mathcal{D}_{\mathbf{ac}}}$ and that of the bottom $\bot_{\mathcal{D}}$ and top of the lattice $\top_{\mathcal{D}}$, which correspond to the extreme points of the lattice of UTP designs, respectively.

C. Demonic choice

The intuition for the definition of demonic choice is related to the possible ways in which the value of ac' can be chosen. In general, this can be described by disjunction like in [2].

Definition 13 (Demonic choice):

$$P \sqcap_{\mathcal{D}_{\mathbf{ac}}} Q \mathrel{\widehat{=}} P \vee Q$$

This corresponds to the greatest lower bound of the lattice. We consider the following example.

Example 3:

$$\begin{aligned}
&(x := 1) \sqcap_{\mathcal{D}_{\mathbf{ac}}} (x := 2) && \{\text{Definition of assignment}\}\\
&= \left(\begin{array}{l} (\mathit{true} \vdash s \oplus (x \mapsto 1) \in ac') \\ \sqcap_{\mathcal{D}_{\mathbf{ac}}} \\ (\mathit{true} \vdash s \oplus (x \mapsto 2) \in ac') \end{array}\right)\\
&\qquad \{\text{Definition of } \sqcap_{\mathcal{D}_{\mathbf{ac}}} \text{ and disjunction of designs}\}\\
&= (\mathit{true} \vdash s \oplus (x \mapsto 1) \in ac' \vee s \oplus (x \mapsto 2) \in ac')
\end{aligned}$$

In this example we have at least two choices for the final value of ac': one has a state where x is 1 and the other has a state where x is 2. The demon can choose any set ac' satisfying either predicate. In this case, the angel is not guaranteed to be able to choose a particular final value for x, since there are no choices in the intersection of all possible choices of ac'.

Following from the closure of **A** with respect to disjunction, the demonic choice operator is also closed. Furthermore, it observes the zero and unit laws regarding the extreme points of the lattice as stated below.

Law 4 ($\sqcap$-$\bot_{\mathcal{D}_{\mathbf{ac}}}$):

$$P \sqcap_{\mathcal{D}_{\mathbf{ac}}} \bot_{\mathcal{D}} = \bot_{\mathcal{D}}$$

Law 5 ($\sqcap$-$\bot_{\mathcal{D}_{\mathbf{ac}}}$):

$$P \sqcap_{\mathcal{D}_{\mathbf{ac}}} \top_{\mathcal{D}} = P$$

This confirms the intuition for demonic choice that if it is possible to abort, then the demon will choose the worst outcome. Furthermore, given the choice between a miraculous program and a design P, the result is P. This indicates the suitability of the definition of demonic choice.

D. Angelic choice

The original theory of designs does not contemplate angelic nondeterminism, and therefore the least upper bound of the lattice of designs, defined as conjunction, does not correspond to angelic choice. In other theories, such as in the predicate transformers model, angelic choice is defined exactly as the dual operator of demonic choice [10]–[12]. The same is applicable for the model of [2], where angelic choice is defined by conjunction, while demonic choice is disjunction. The definition adopted in our model is also conjunction of designs.

Definition 14 (Angelic choice):

$$P \sqcup_{\mathcal{D}_{\mathbf{ac}}} Q \mathrel{\widehat{=}} P \wedge Q$$

To provide the intuition for this definition we consider the following example.

Example 4:

$$\begin{aligned}
&\left(\begin{array}{l} ((x \mapsto 1) \notin ac' \vdash (x \mapsto 1) \in ac') \\ \sqcup_{\mathcal{D}_{\mathbf{ac}}} \\ (\mathit{true} \vdash (x \mapsto 2) \in ac') \end{array}\right)\\
&\qquad \{\text{Definition of } \sqcup_{\mathcal{D}_{\mathbf{ac}}}\}\\
&= \left(\begin{array}{l} (x \mapsto 1) \notin ac' \vee \mathit{true} \\ \vdash \\ \left(\begin{array}{l} (x \mapsto 1) \notin ac' \Rightarrow (x \mapsto 1) \in ac' \\ \wedge \\ \mathit{true} \Rightarrow (x \mapsto 2) \in ac' \end{array}\right) \end{array}\right)\\
&\qquad \{\text{Predicate calculus}\}\\
&= (\mathit{true} \vdash (x \mapsto 1) \in ac' \wedge (x \mapsto 2) \in ac')
\end{aligned}$$

It considers the angelic choice between a design that assigns 1 to the only program variable x, but does not necessarily terminate, and a design that assigns 2 to x but terminates. The result is a program that terminates and, for every set of final states, there is the possibility for the angel to choose the assignment of the value 1 or 2 to x.

In general, and since angelic choice corresponds to the least upper bound, the angelic choice of a design P and the top of the lattice $\top_{\mathcal{D}}$ is also $\top_{\mathcal{D}}$.

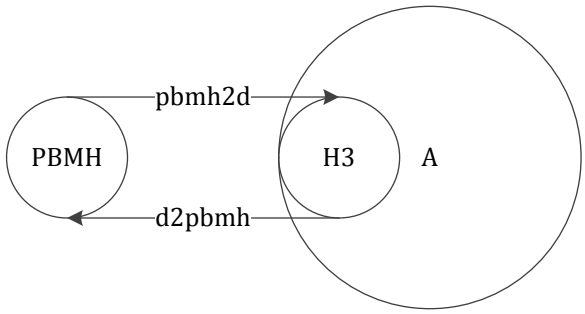

Fig. 1. Link between the theories.

Law 6 ($\sqcup_{\mathcal{D}\mathbf{ac}} - \top_{\mathcal{D}}$): Provided P is a design.

$$P \sqcup_{\mathcal{D}\mathbf{ac}} \top_{\mathcal{D}} = \top_{\mathcal{D}}$$

Proof:

$$\begin{array}{ll} P \sqcup_{\mathcal{D}\mathbf{ac}} \top_{\mathcal{D}} & \{\text{Definition of } \sqcup_{\mathcal{D}\mathbf{ac}} \text{ and } \top_{\mathcal{D}}\} \\ = P \wedge \neg\, ok & \{\text{Definition of design}\} \\ = (\neg\, P^f \vdash P^t) \wedge \neg\, ok & \{\text{Definition of design}\} \\ = ((ok \wedge \neg\, P^f) \Rightarrow (P^t \wedge ok')) \wedge \neg\, ok & \{\text{Predicate calculus}\} \\ = (\neg\, ok \vee P^f \vee (P^t \wedge ok')) \wedge \neg\, ok & \{\text{Predicate calculus: absorption law}\} \\ = \neg\, ok & \{\text{Definition of } \top_{\mathcal{D}}\} \\ = \top_{\mathcal{D}} & \end{array}$$

■

Law 7 ($\sqcup_{\mathcal{D}\mathbf{ac}} - \bot_{\mathcal{D}}$):

$$P \sqcup_{\mathcal{D}\mathbf{ac}} \bot_{\mathcal{D}} = P$$

Furthermore, the angelic choice between a design P and the bottom of the lattice $\bot_{\mathcal{D}}$ is also P. These laws indicate the suitability of the angelic choice operator.

V. Linking theories

If we restrict the theory of designs that we propose to **H3**-healthy designs, then we can relate it with the theory in [2]. For this purpose we define a pair of linking functions between the models that we illustrate in Figure 1. The function *pbmh2d* maps from **PBMH**-healthy predicates into the subset of our theory while *d2pbmh* maps in the opposite direction.

The major difference between the theory in [2] and our new theory is the use of the auxiliary variables ok and ok' for capturing termination. In addition, as discussed earlier, instead of considering all input program variables, we have a single initial state denoted by s that encapsulates them as record components. Furthermore, all sets of final states in our theory have undashed variables, rather than dashed variables. The relationship between the sets of states in both models can, therefore, be formalized by the functions *ac2acdash* and its inverse *acdash2ac*.

Definition 15 (acdash-to-ac):

$$acdash2ac(ss) = \left\{ \begin{array}{l} s_0 : S_{in\alpha}, s_1 : S_{out\alpha} \\ \mid\ s_1 \in ss \wedge \\ \mid\ (\bigwedge x : \alpha P \bullet s_0.x = s_1.x') \bullet s_0 \end{array} \right\}$$

$$ac2acdash(zz) = \left\{ \begin{array}{l} z_0 : S_{in\alpha}, z_1 : S_{out\alpha} \\ \mid\ z_0 \in ss \wedge \\ \mid\ (\bigwedge x : \alpha P \bullet z_0.x = z_1.x') \bullet z_1 \end{array} \right\}$$

The function *acdash2ac* maps a set of angelic choices ss whose record components are dashed variables into a set whose record components are undashed. This is achieved by considering every state s_1 in ss and every state s_0, such that s_0 is a state on the undashed variables of predicate P and whose components are exactly the same as those in s_1, except that those in s_1 are dashed. We use $S_{in\alpha}$ to denote the set of input variables for a given program while $S_{out\alpha}$ denotes the set of output variables.

A. From designs to PBMH predicates

The first linking function of interest is *d2pbmh* that maps from designs that are **A** and **H3**-healthy into the theory of [2].

Definition 16:

$$d2pbmh : \mathbf{A} \nrightarrow \mathbf{PBMH}$$

$$d2pbmh(P) \mathrel{\widehat{=}} \left(\begin{array}{l} \exists\, ac_0 \bullet (\neg\, P^f \Rightarrow P^t)[ac_0/ac'][in\alpha/s] \\ \wedge\ ac2acdash(ac_0) \subseteq ac' \end{array} \right)$$

For a design P we consider both its pre and postconditions directly. This is sufficient since we require ok to be *true* and hide ok'. The substitution of $in\alpha$ for s corresponds to the substitution of every occurrence of a record component $s.x$ with x, where x is an input program variable. Finally, we substitute the temporary variable ac_0 for ac' in P. This allows us to relate the set of final states ac_0 with ac' by applying *ac2acdash*.

B. From PBMH predicates to designs

Definition 17:

$$pbmh2d : \mathbf{PBMH} \nrightarrow \mathbf{A}$$

$$pbmh2d(P) \mathrel{\widehat{=}} \left(\begin{array}{l} \neg\, P[\emptyset/ac'][s/in\alpha] \\ \vdash \\ \exists\, ac_0 \bullet P[ac_0/ac'][s/in\alpha] \\ \wedge\ acdash2ac(ac_0) \subseteq ac' \end{array} \right)$$

The definition yields a design whose precondition guarantees successful termination, the postcondition follows the same idea explored in the definition of *d2pbmh*. Every input program variable x in $in\alpha$ is substituted with $s.x$, where s is the initial state, and ac_0 is related to ac' by application of *acdash2ac*.

In the model of [2], the possibility of non termination occurs when ac' is the empty set. Therefore the negation of this predicate can be taken as a precondition.

C. Isomorphism

Together the linking functions *d2pbmh* and *pbmh2d* are a bijection. This result is established by the following theorems, whose proofs are available in [18].

Theorem 2: Provided P is $\mathbf{A} \circ \mathbf{H3}$-healthy.

$$pbmh2d \circ d2pbmh(P) = P$$

Theorem 3: Provided P is **PBMH**-healthy.

$$d2pbmh \circ pbmh2d(P) = P$$

While our theory of designs has a different alphabet, it is reassuring that the subset of our theory that is **H3**-healthy is in exact correspondence with the UTP theory of [2]. This indicates that our results are consistent with the existing model.

VI. Conclusion

Angelic nondeterminism is a useful specification construct. It has been extensively studied in the context of theories of correctness for sequential programs [2], [10]–[12]. In particular, in the universally monotonic predicate transformers model it is defined precisely as the dual of demonic nondeterminism.

The characterisation of angelic nondeterminism in a relational setting such as the UTP, is however, more challenging [2]. Similarly to [2] the theory that we propose is also a theory of non-homogeneous relations. As a consequence sequential composition is not relational composition. Despite the unconventional definition that we propose, the results indicate that it is consistent with a theory of designs. We have also shown that the subset of **H3**-designs of our theory is just as expressive as that of [2]. Moreover, the fact that we can describe sequential composition in a way similar to that adopted in the theory of reactive designs, by lifting the sequential composition operator of the theory in [2] is both pleasing and reassuring.

Other works regarding angelic nondeterminism include those of Morris and Tyrrel [19]–[22], and Hesselink [23], who have pursued the modelling of both types of nondeterminism at the expression or term level. Their focus is on functional languages. Tyrrell et al. [24] have attempted an axiomatization for an algebra resembling CSP where external choice is referred to as "angelic choice", however this is quite different from the semantics of external choice in standard CSP [8].

In [25] a modelling approach for the verification of implementations of control systems based on the notation *Circus Time* [6] is proposed that uses angelic nondeterminism. This is a notation whose semantics are defined using the UTP. The development of a practical verification for such an approach would require a theory of reactive processes with angelic nondeterminism. As far as we know, no definitive treatment of angelic nondeterminism has been provided in the context of process algebras such as CSP.

It is our objective to consider a theory of reactive programs using the theory of designs that we propose here. This will allow us to explore the consequences of angelic nondeterminism with respect to reactive programs.

Acknowledgment

The work presented here is funded by the EPSRC, UK.

References

[1] C. A. R. Hoare and H. Jifeng, *Unifying Theories of Programming*. Prentice Hall International Series in Computer Science, 1998.

[2] A. Cavalcanti, J. Woodcock, and S. Dunne, "Angelic nondeterminism in the unifying theories of programming," *Formal Aspects of Computing*, vol. 18, pp. 288–307, 2006.

[3] T. Santos, A. Cavalcanti, and A. Sampaio, "Object-Orientation in the UTP," in *Unifying Theories of Programming*, ser. Lecture Notes in Computer Science, S. Dunne and B. Stoddart, Eds. Springer Berlin / Heidelberg, 2006, vol. 4010, pp. 18–37.

[4] W. Harwood, A. Cavalcanti, and J. Woodcock, "A Theory of Pointers for the UTP," in *Theoretical Aspects of Computing - ICTAC 2008*, ser. Lecture Notes in Computer Science, J. Fitzgerald, A. Haxthausen, and H. Yenigun, Eds. Springer Berlin / Heidelberg, 2008, vol. 5160, pp. 141–155.

[5] A. Sherif and J. He, "Towards a Time Model for *circus*," in *Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering*, ser. ICFEM '02. London, UK, UK: Springer-Verlag, 2002, pp. 613–624.

[6] A. Sherif, "A Framework for Specification and Validation of Real-Time Systems using *Circus Actions*," Ph.D. dissertation, Center of Informatics - Federal University of Pernambuco, Brazil, 2006. [Online]. Available: http://www.cs.york.ac.uk/circus/publications/papers/06-sherif.pdf

[7] K. Wei, J. Woodcock, and A. Cavalcanti, "New *Circus Time*," University of York, Tech. Rep., April 2013. [Online]. Available: http://www.cs.york.ac.uk/circus/publications/techreports/reports/Circus%20Time.pdf

[8] A. W. Roscoe, *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[9] A. Cavalcanti and J. Woodcock, "A Tutorial Introduction to CSP in *Unifying Theories of Programming*," in *Refinement Techniques in Software Engineering*, ser. Lecture Notes in Computer Science, A. Cavalcanti, A. Sampaio, and J. Woodcock, Eds. Springer Berlin / Heidelberg, 2006, vol. 3167, pp. 220–268.

[10] R. Back and J. Wright, *Refinement calculus: a systematic introduction*, ser. Graduate texts in computer science. Springer, 1998.

[11] J. M. Morris, "A theoretical basis for stepwise refinement and the programming calculus," *Sci. Comput. Program.*, vol. 9, pp. 287–306, December 1987.

[12] C. Morgan, *Programming from specifications*. Prentice Hall, 1994.

[13] R. Back and J. von Wright, "Combining angels, demons and miracles in program specifications," *Theoretical Computer Science*, vol. 100, no. 2, pp. 365 – 383, 1992.

[14] I. Rewitzky, "Binary Multirelations," in *Theory and Applications of Relational Structures as Knowledge Instruments*, ser. Lecture Notes in Computer Science, H. de Swart, E. Orlowska, G. Schmidt, and M. Roubens, Eds. Springer Berlin / Heidelberg, 2003, vol. 2929, pp. 1964–1964.

[15] C. E. Martin, S. A. Curtis, and I. Rewitzky, "Modelling Nondeterminism," in *MPC, volume 3125 of LNCS*. Springer, 2004, pp. 228–251.

[16] J. Woodcock and A. Cavalcanti, "A Tutorial Introduction to Designs in Unifying Theories of Programming," in *Integrated Formal Methods*, ser. Lecture Notes in Computer Science, E. Boiten, J. Derrick, and G. Smith, Eds. Springer Berlin / Heidelberg, 2004, vol. 2999, pp. 40–66.

[17] A. Cavalcanti and J. Woodcock, "Angelic Nondeterminism and Unifying Theories of Programming," University of Kent, Tech. Rep., 2004. [Online]. Available: http://kar.kent.ac.uk/14151/

[18] P. Ribeiro, "Designs with angelic nondeterminism," University of York, Technical Report, February 2013. [Online]. Available: http://www-users.cs.york.ac.uk/pfr/dac/designs-dac.pdf

[19] J. M. Morris and M. Tyrrell, "Terms with unbounded demonic and angelic nondeterminacy," *Science of Computer Programming*, vol. 65, no. 2, pp. 159 – 172, 2007.

[20] J. Morris and M. Tyrrell, "Dual unbounded nondeterminacy, recursion, and fixpoints," *Acta Informatica*, vol. 44, pp. 323–344, 2007.

[21] J. M. Morris and M. Tyrrell, "Dually nondeterministic functions," *ACM Trans. Program. Lang. Syst.*, vol. 30, pp. 34:1–34:34, October 2008.

[22] J. Morris and M. Tyrrell, "Modelling higher-order dual nondeterminacy," *Acta Informatica*, vol. 45, pp. 441–465, 2008. [Online]. Available: http://dx.doi.org/10.1007/s00236-008-0076-1

[23] W. H. Hesselink, "Alternating states for dual nondeterminism in imperative programming," *Theoretical Computer Science*, vol. 411, no. 22-24, pp. 2317 – 2330, 2010.

[24] M. Tyrrell, J. Morris, A. Butterfield, and A. Hughes, "A Lattice-Theoretic Model for an Algebra of Communicating Sequential Processes," in *Theoretical Aspects of Computing - ICTAC 2006*, ser. Lecture Notes in Computer Science, K. Barkaoui, A. Cavalcanti, and A. Cerone, Eds. Springer Berlin / Heidelberg, 2006, vol. 4281, pp. 123–137.

[25] A. Cavalcanti and M. Alexandre, "Simulink in *Circus Time*."

APPENDIX

Lemma 3: Provided P satisfies **PBMH**.

$$P \;;_{\mathcal{A}}\; (Q \wedge ok') = (P \;;_{\mathcal{A}}\; \mathit{false}) \vee ((P \;;_{\mathcal{A}}\; Q) \wedge ok')$$

Proof of Lemma 2:

$$\begin{array}{ll}
& \mathbf{A0} \circ \mathbf{A1}(P) \\
& \qquad \{\text{Definition of design}\} \\
= & \mathbf{A0} \circ \mathbf{A1}(\neg P^f \vdash P^t) \\
& \qquad \{\text{Definition of } \mathbf{A1}\} \\
= & \mathbf{A0}(\neg\, \mathbf{PBMH}(P^f) \vdash \mathbf{PBMH}(P^t)) \\
& \qquad \{\text{Definition of } \mathbf{A0}\} \\
= & (\neg\, \mathbf{PBMH}(P^f) \vdash \mathbf{PBMH}(P^t) \wedge ac' \neq \emptyset) \\
& \qquad \{ac' \neq \emptyset \text{ satisfies } \mathbf{PBMH}\} \\
= & (\neg\, \mathbf{PBMH}(P^f) \vdash \mathbf{PBMH}(P^t) \wedge \mathbf{PBMH}(ac' \neq \emptyset)) \\
& \qquad \{\text{Closure of } \mathbf{PBMH} \text{ w.r.t. conjunction}\} \\
= & (\neg\, \mathbf{PBMH}(P^f) \vdash \mathbf{PBMH}(\mathbf{PBMH}(P^t) \wedge \mathbf{PBMH}(ac' \neq \emptyset))) \\
& \qquad \{ac' \neq \emptyset \text{ satisfies } \mathbf{PBMH}\} \\
= & (\neg\, \mathbf{PBMH}(P^f) \vdash \mathbf{PBMH}(\mathbf{PBMH}(P^t) \wedge ac' \neq \emptyset)) \\
& \qquad \{\text{Assumption: } P^t \text{ satisfies } \mathbf{PBMH}\} \\
= & (\neg\, \mathbf{PBMH}(P^f) \vdash \mathbf{PBMH}(P^t \wedge ac' \neq \emptyset)) \\
& \qquad \{\text{Definition of } \mathbf{A1}\} \\
= & \mathbf{A1}(\neg P^f \vdash P^t \wedge ac' \neq \emptyset) \\
& \qquad \{\text{Definition of } \mathbf{A0}\} \\
= & \mathbf{A1} \circ \mathbf{A0}(\neg P^f \vdash P^t) \\
& \qquad \{\text{Definition of design}\} \\
= & \mathbf{A1} \circ \mathbf{A0}(P)
\end{array}$$

■

Proof of Theorem 1:

$$\begin{array}{ll}
& (P_0 \vdash P_1) \;;_{\mathcal{D}\mathbf{ac}}\; (Q_0 \vdash Q_1) \\
& \qquad \{\text{Definition of design}\} \\
= & \left(\begin{array}{l} ((ok \wedge P_0) \Rightarrow (P_1 \wedge ok')) \\ ;_{\mathcal{D}\mathbf{ac}} \\ ((ok \wedge Q_0) \Rightarrow (Q_1 \wedge ok')) \end{array} \right) \\
& \qquad \{\text{Definition of } ;_{\mathcal{D}\mathbf{ac}}\} \\
= & \exists\, ok_0 \bullet \left(\begin{array}{l} ((ok \wedge P_0) \Rightarrow (P_1 \wedge ok'))[ok_0/ok'] \\ ;_{\mathcal{A}} \\ ((ok \wedge Q_0) \Rightarrow (Q_1 \wedge ok'))[ok_0/ok] \end{array} \right) \\
& \qquad \{\text{Substitution}\} \\
= & \exists\, ok_0 \bullet \left(\begin{array}{l} ((ok \wedge P_0) \Rightarrow (P_1 \wedge ok_0)) \\ ;_{\mathcal{A}} \\ ((ok_0 \wedge Q_0) \Rightarrow (Q_1 \wedge ok')) \end{array} \right) \\
& \qquad \{\text{Case-analysis on } ok_0\} \\
= & \left(\begin{array}{l} (((ok \wedge P_0) \Rightarrow P_1) \;;_{\mathcal{A}}\; (Q_0 \Rightarrow (Q_1 \wedge ok'))) \\ \vee \\ ((\neg\, (ok \wedge P_0)) \;;_{\mathcal{A}}\; \mathit{true}) \end{array} \right) \\
& \qquad \{\text{Predicate calculus}\} \\
= & \left(\begin{array}{l} ((\neg\, ok \vee \neg\, P_0 \vee P_1) \;;_{\mathcal{A}}\; (Q_0 \Rightarrow (Q_1 \wedge ok'))) \\ \vee \\ ((\neg\, ok \vee \neg\, P_0) \;;_{\mathcal{A}}\; \mathit{true}) \end{array} \right) \\
& \qquad \{\text{Right-distributivity of } ;_{\mathcal{A}}\} \\
= & \left(\begin{array}{l} (\neg\, ok \;;_{\mathcal{A}}\; (Q_0 \Rightarrow (Q_1 \wedge ok'))) \\ \vee \\ (\neg\, P_0 \;;_{\mathcal{A}}\; (Q_0 \Rightarrow (Q_1 \wedge ok'))) \\ \vee \\ (P_1 \;;_{\mathcal{A}}\; (Q_0 \Rightarrow (Q_1 \wedge ok'))) \\ \vee \\ (\neg\, ok \;;_{\mathcal{A}}\; \mathit{true}) \vee (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \end{array} \right) \\
& \qquad \{\text{Property of } ;_{\mathcal{A}} \ (ac' \text{ is not free}) \text{ and predicate calculus}\} \\
= & \left(\begin{array}{l} \neg\, ok \vee (\neg\, P_0 \;;_{\mathcal{A}}\; (Q_0 \Rightarrow (Q_1 \wedge ok'))) \\ \vee \\ (P_1 \;;_{\mathcal{A}}\; (Q_0 \Rightarrow (Q_1 \wedge ok'))) \\ \vee \\ (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \end{array} \right) \\
& \qquad \{\text{Assumption: } P \text{ is } \mathbf{A}\text{-healthy, and left distributivity of } ;_{\mathcal{A}}\} \\
= & \left(\begin{array}{l} \neg\, ok \vee (\neg\, P_0 \;;_{\mathcal{A}}\; ((Q_0 \Rightarrow (Q_1 \wedge ok')) \vee \mathit{true})) \\ \vee \\ (P_1 \;;_{\mathcal{A}}\; (Q_0 \Rightarrow (Q_1 \wedge ok'))) \end{array} \right) \\
& \qquad \{\text{Predicate calculus}\} \\
= & \neg\, ok \vee (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \vee (P_1 \;;_{\mathcal{A}}\; (\neg\, Q_0 \vee (Q_1 \wedge ok'))) \\
& \qquad \{\text{Assumption: } P \text{ is } \mathbf{A}\text{-healthy, and left distributivity of } ;_{\mathcal{A}}\} \\
= & \left(\begin{array}{l} \neg\, ok \vee (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \vee (P_1 \;;_{\mathcal{A}}\; \neg\, Q_0) \\ \vee \\ (P_1 \;;_{\mathcal{A}}\; (Q_1 \wedge ok')) \end{array} \right) \\
& \qquad \{\text{Assumption: } P \text{ is } \mathbf{A}\text{-healthy, and Lemma 3}\} \\
= & \left(\begin{array}{l} \neg\, ok \vee (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \vee (P_1 \;;_{\mathcal{A}}\; \neg\, Q_0) \\ \vee\, (P_1 \;;_{\mathcal{A}}\; \mathit{false}) \vee ((P_1 \;;_{\mathcal{A}}\; Q_1) \wedge ok') \end{array} \right) \\
& \qquad \{\text{Assumption: } P \text{ is } \mathbf{A}\text{-healthy, and left distributivity of } ;_{\mathcal{A}}\} \\
= & \left(\begin{array}{l} \neg\, ok \vee (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \vee \\ (P_1 \;;_{\mathcal{A}}\; (\neg\, Q_0 \vee \mathit{false})) \vee ((P_1 \;;_{\mathcal{A}}\; Q_1) \wedge ok') \end{array} \right) \\
& \qquad \{\text{Predicate calculus}\} \\
= & \left(\begin{array}{l} \neg\, ok \vee (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \vee (P_1 \;;_{\mathcal{A}}\; \neg\, Q_0) \\ \vee\, ((P_1 \;;_{\mathcal{A}}\; Q_1) \wedge ok') \end{array} \right) \\
& \qquad \{\text{Predicate calculus}\} \\
= & \left(\begin{array}{l} ok \wedge \neg\, (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \\ \wedge\, \neg\, (P_1 \;;_{\mathcal{A}}\; \neg\, Q_0) \end{array} \right) \Rightarrow ((P_1 \;;_{\mathcal{A}}\; Q_1) \wedge ok') \\
& \qquad \{\text{Definition of design}\} \\
= & (\neg\, (\neg\, P_0 \;;_{\mathcal{A}}\; \mathit{true}) \wedge \neg\, (P_1 \;;_{\mathcal{A}}\; \neg\, Q_0) \vdash P_1 \;;_{\mathcal{A}}\; Q_1)
\end{array}$$

■