Semantics, Orderings and Recursion in the Weakest Precondition Calculus

Marcello Bonsangue*
CWI,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands.
Email: `marcello@cwi.nl`.

Joost N. Kok
Utrecht University, Department of Computer Science,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.
Email: `joost@cs.ruu.nl`.

ABSTRACT An extension of Dijkstra's guarded command language is studied, including sequential composition, demonic choice and a backtrack operator. To guide the intuition about this language we give an operational semantic that relates the initial states with possible outcome of the computations. Next we consider three orderings on this language: a refinement ordering defined by Back, a new deadlock ordering, and an approximation ordering of Nelson. The deadlock ordering is in between the two other orderings. All operators are monotonic in Nelson's ordering, but backtracking is not monotonic in Back's ordering and sequential composition is not monotonic for the deadlock ordering. At first sight recursion can only be added using Nelson's ordering. By extending the fixed point theory we show that, under certain circumstances, least fixed points for non monotonic functions can be obtained by iteration from the least element. This permits us the addition of recursion even using Back's ordering or the deadlock ordering. Furthermore, we give a semantic characterization of the three orderings above by extending the well known duality theory between predicate transformers and Smyth's powerdomain.

Keywords weakest preconditions, predicate transformers, refinement, deadlock, backtracking, recursion, fixed points, fixed point transformations, Smyth powerdomain, Egli-Milner powerdomain.

CONTENTS

*The research of this author was supported by a grant of the Universita' degli Studi di Milano, Italy.

1 Introduction

The weakest precondition calculus of Dijkstra identifies statements in the guarded command language with weakest precondition predicate transformers (see [Dij76]). The language was extended to use it as a vehicle for program refinement. Specification constructs were added and a refinement ordering was defined. This approach was introduced in [Bac78, Bac80] and is suited for refinement (see [BvW90, Bac90] and also [MRG88, Mor87]). The refinement ordering can be used to add recursion to the language, but not in a fully compositional way. For example, for each set of guards there is a different conditional command.

Recursion was added in a fully compositional way by Nelson in [Nel87]: the guarded command language was embedded in a language with sequential composition, demonic choice and a backtrack operator in which the operators can be used freely. An ordering is given for which the operators are all monotonic. This ordering is an approximation ordering of the kind used in denotational semantics and does not seem to be suited for refinement. It is defined with the additional notion of weakest liberal preconditions.

Our starting point is the language of [Nel87]. In this language we also have a form of infinite behaviour (a loop construct) and atomic actions that can deadlock (to initiate backtracking). Then we consider three orderings; besides the orderings of Back and Nelson we define a new ordering in between. It is called deadlock ordering because it preserves deadlocks as can be seen from the semantic characterization of the deadlock ordering. In terms of refinement: a normal (non-miraculous) terminating statement is not refined by a miracle in the deadlock ordering.

Only Nelson's ordering is monotonic with respect to all three operators, while the backtrack operator is not monotonic with respect to Back's ordering and the sequential composition is not monotonic for the deadlock ordering. At first sight only Nelson's ordering seems to be suited to add recursion to the full language. But the fact that for Nelson's ordering all the operators are monotonic implies that also recursion can be added with the other two orderings.

In order to show this we extend the fixed point theory. It is well known that a monotone and continuous function from a complete partial order to itself has least fixed point that can be obtained by iteration from the least element. This result was extended at first by Hitchcock and Park [HP72] showing that for a function from a complete partial order to itself is enough to be monotone in order to have a lest fixed point. Then Apt and Ploktin [AP86] have shown that the least fixed point property can be transferred, via a commutative diagram, to monotone functions from a partial order to itself. Finally, in [BK92] we show that the least fixed point property can be transferred, via a commutative diagram, also to functions (even non monotone) from a partial order to itself. Here we give a theorem that uses only part of the results given in [BK92], but this theorem is enough to imply that for both Back's and the deadlock ordering the standard operator associated to a declaration of recursive procedures has a least fixed point that can be obtained by iteration from the least element. It also gives the correct result because it is related to the least fixed point with respect to Nelson's ordering.

Moreover we provide a semantic characterization of the three orderings based on a semantic model for the language that relates initial states to possible outcomes of the computation. We start from the duality theory connecting the discrete version of the Smyth powerdomain [Smy78] and the Dijkstra's predicate transformers [Wan77, Plo79, Smy83, Bes83, AP86]. The presence of a backtrack operator in our language justifies the introduction of two different versions of the Smyth powerdomain in which a constant representing the deadlock is added in two different way. We extend the duality theory described above to these two versions of the Smyth powerdomain giving in this way a semantic characterization for the Back and the deadlock orderings. A similar result is also proved for the Egli-Milner powerdomain showing its relationship with the Nelson's predicate transformers.

For reason of space, almost all the proofs are omitted; they can be found in [BK92].

2 Language and Semantics

We first introduce the language. We use the notation $(d \in)Dom$ to introduce the domain Dom and a typical element d of this domain. Function application is denoted by . and associates to the left, that is $f.g.x = (f.g).x$.

Let $(v \in)Var$ be a set of variables, let $(t \in)IExp$ be a set of integer expressions, and let $(b \in)BExp$ be a set of boolean expressions. Then the set $(S \in)Stat$ is defined by

$$S \ ::= \ v := t \mid b \rightarrow \ \mid loop \mid S_1; S_2 \mid S_1 \Box S_2 \mid S_1 \Diamond S_2.$$

This language has three operators: the sequential composition ; , the demonic choice $\Box$, and the backtrack operator $\Diamond$.

The backtrack operator backtracks to its second component if its first component deadlocks. The only atomic action that can deadlock is $b \rightarrow$: it deadlocks in a state in which the boolean expression b does not evaluate to true. A form of infinite behaviour (the *loop*-statement) is added to the language to distinguish different orderings on the language. A similar language is studied in [Nel87]: the only difference is that we have split actions as in [Hes89] in the sense that we consider as atomic actions both the assignment actions $v := t$ and the test actions $b \rightarrow$.

Dijkstra's guarded command language [Dij76] can be seen as a subset of this language, except for the **do** – **od**-construct which will be handled when we add recursion. For example, the conditional command **if** $b_1 \rightarrow S_1 \Box b_2 \rightarrow S_2$ **fi** can be expressed by the statement $(b_1 \rightarrow\ ;\ S_1 \Box b_2 \rightarrow\ ;\ S_2)\Diamond loop \in Stat$. More general derived statements are $skip = \mathbf{true} \rightarrow$, $abort = loop$, $magic = \mathbf{false} \rightarrow$, and **if** S **fi** $= S\Diamond loop$.

Next we give an operational semantic model that relates initial states with possible outcomes of the computation. A state is a function that yields an integer for each variable in $(v \in)\, Var$, thus the set of states $(\sigma \in)\Sigma$ is given by $\Sigma = Var \rightarrow N$. Also, we assume that we can consider integer expressions t as functions that given a state σ yield an integer $t.\sigma$. The same applies to boolean expressions b.

We introduce a set of extended statements $(m \in)\overline{Stat}$ to treat backtracking in a transition system:

$$m \ ::= \ S \mid m_1 \triangle (m_2, \sigma)$$

where $S \in Stat$ and $\sigma \in \Sigma$. After the next definition we give some more explanation.

Definition 2.1 *Let $Conf = (\overline{Stat} \cup \{E\}) \times (\Sigma \cup \{\delta\})$ be a set of configurations, and define a transition relation $\longrightarrow \subseteq Conf \times Conf$ to be the least relation satisfying the following axioms and rules:*

$$\langle v := t, \sigma \rangle \longrightarrow \langle E, \sigma[t.\sigma/v] \rangle$$

$$\langle b \rightarrow, \sigma \rangle \longrightarrow \langle E, \delta \rangle \ \text{ if not } b.\sigma \qquad\qquad \langle b \rightarrow, \sigma \rangle \longrightarrow \langle E, \sigma \rangle \ \text{ if } b.\sigma$$

$$\langle loop, \sigma \rangle \longrightarrow \langle loop, \sigma \rangle$$

$$\frac{\langle m_1, \sigma \rangle \longrightarrow \langle E, \delta \rangle}{\langle m_1;\, m_2, \sigma \rangle \longrightarrow \langle E, \delta \rangle} \qquad\qquad \frac{\langle m_1, \sigma \rangle \longrightarrow \langle m_1' | E, \sigma' \rangle}{\langle m_1;\, m_2, \sigma \rangle \longrightarrow \langle m_1';\, m_2 | m_2, \sigma' \rangle}$$

$$\frac{\langle m_1, \sigma \rangle \longrightarrow \langle E, \delta \rangle \ \wedge \ \langle m_2, \sigma \rangle \longrightarrow \langle E, \delta \rangle}{\langle m_1 \Box m_2, \sigma \rangle \longrightarrow \langle E, \delta \rangle}$$

$$\frac{\langle m_1, \sigma \rangle \longrightarrow \langle m_1' | E, c' \rangle}{\langle m_1 \Box m_2, \sigma \rangle \longrightarrow \langle m_1' | E, \sigma' \rangle} \qquad\qquad \frac{\langle m_1, \sigma \rangle \longrightarrow \langle m_1' | E, \sigma' \rangle}{\langle m_2 \Box m_1, \sigma \rangle \longrightarrow \langle m_1' | E, \sigma' \rangle}$$

$$\frac{\langle m_1, \sigma \rangle \longrightarrow \langle E, \delta \rangle \ \wedge \ \langle m_2, \sigma \rangle \longrightarrow \langle E, \delta \rangle}{\langle m_1 \Diamond m_2, \sigma \rangle \longrightarrow \langle E, \delta \rangle}$$

$$\frac{\langle m_1, \sigma \rangle \longrightarrow \langle E, \delta \rangle \ \wedge \ \langle m_2, \sigma \rangle \longrightarrow \langle m_2' | E, \sigma' \rangle}{\langle m_1 \Diamond m_2, \sigma \rangle \longrightarrow \langle m_2' | E, \sigma' \rangle} \qquad\qquad \frac{\langle m_1, \sigma \rangle \longrightarrow \langle m_1' | E, \sigma' \rangle}{\langle m_1 \Diamond m_2, \sigma \rangle \longrightarrow \langle m_1' \triangle (m_2, \sigma) | E, \sigma' \rangle}$$

$$\frac{\langle m_1, \sigma \rangle \longrightarrow \langle E, \delta \rangle \ \wedge \ \langle m_2, \sigma' \rangle \longrightarrow \langle E, \delta \rangle}{\langle m_1 \triangle (m_2, \sigma'), \sigma \rangle \longrightarrow \langle E, \delta \rangle}$$

$$\frac{\langle m_1, \sigma \rangle \longrightarrow \langle E, \delta \rangle \ \wedge \ \langle m_2, \sigma' \rangle \longrightarrow \langle m_2' | E, \sigma'' \rangle}{\langle m_1 \triangle (m_2, \sigma'), \sigma \rangle \longrightarrow \langle m_2' | E, \sigma'' \rangle} \qquad\qquad \frac{\langle m_1, \sigma \rangle \longrightarrow \langle m_1' | E, \sigma' \rangle}{\langle m_1 \triangle (m_2, \sigma''), \sigma \rangle \longrightarrow \langle m_1' \triangle (m_2, \sigma'') | E, \sigma' \rangle}$$

In the definition above $\sigma[t.\sigma/v]$ denotes the state

$$(\sigma[t.\sigma/v]).v' = \begin{cases} t.\sigma & \text{if } v = v' \\ \sigma.v & \text{otherwise.} \end{cases}$$

Furthermore $\langle m_1 | E, \sigma \rangle$ is an abbreviation for the two alternative configurations $\langle m_1, \sigma \rangle$ and $\langle E, \sigma \rangle$. Intuitively, $\langle m_1, \sigma \rangle \longrightarrow \langle m_1', \sigma' \rangle$ states that one step of execution of the statement m_1 in the state σ leads to a state σ' with m_1' being the remainder of m_1 to be executed.

Definition 2.2 *We say that m can diverge from σ, denoted by $(m, \sigma) \uparrow$, if there exists an infinite sequence of configuration c_i such that*

$$(\forall i \geq 0 \ : \ c_i \longrightarrow c_{i+1})$$

where $c_0 = \langle m, \sigma \rangle$. Furthermore, by $c_0 \longrightarrow^* c_n'$ we denote that there exists a finite sequence of configuration c_i such that

$$c_0 \longrightarrow c_1 \longrightarrow \cdots \longrightarrow c_{n-1} \longrightarrow c_n'$$

For each statement in *Stat* we can now define its operational semantics:

Definition 2.3 *Let the function $Op : Stat \rightarrow (\Sigma \rightarrow \mathcal{P}.\Sigma \cup \Sigma_\perp)^1$ defined by:*

$$Op.S.\sigma = \begin{cases} \Sigma_\perp & \textit{if } (S, \sigma) \uparrow \\ \{\sigma' | \langle S, \sigma \rangle \longrightarrow^* \langle E, \sigma' \rangle \ \} & \textit{otherwise} \end{cases}$$

The definition of the function *Op* explains why $\Box$ is called demonic choice: if there is the possibility of infinite behaviour (S can diverge) then it will be chosen. Next we discuss the backtrack operator $\Diamond$. If we execute the statement $S_1 \Diamond S_2$ in a state σ then we look if we can do a step from S_1 (that possibly changes σ say in σ') and we remember the starting state σ changing $\Diamond$ in $\triangle$. If this computation deadlocks at a later stage, then we still have the alternative S_2 left reinstalling the state σ.

As a second step we define the weakest precondition semantics and relate it to the model *Op*. Let $\mathbf{B} = \{tt, ff\}$ be the boolean set and $(P, Q \in) Pred = \Sigma \rightarrow \mathbf{B}$ be predicates.

Definition 2.4 *(weakest preconditions) Let $wp \ : \ Stat \rightarrow (Pred \rightarrow Pred)$ be defined as follows:*

$$\begin{array}{ll} wp.b \rightarrow .Q = b \Rightarrow Q & wp.S_1; S_2.Q = wp.S_1.(wp.S_2.Q) \\ wp.v := t.Q = Q[t/v] & wp.S_1 \Box S_2.Q = wp.S_1.Q \wedge wp.S_2.Q \\ wp.loop.Q = \mathbf{false} & wp.S_1 \Diamond S_2.Q = wp.S_1.Q \wedge (wp.S_1.\mathbf{false} \Rightarrow wp.S_2.Q). \end{array}$$

In this definition $Q[t/v]$ denotes syntactic substitution in Q of t for v. It is not difficult to prove that for any statement S the predicate transformer $wp.S$ is monotonic with respect to $\Rightarrow$: we have that if $P \Rightarrow Q$ then $wp.S.P \Rightarrow wp.S.Q$.

The following theorem relates the weakest precondition semantics with the operational semantics in the same way as in [Bak80]; at first generalize predicates P from Σ to $(\mathcal{P}.\Sigma \cup \Sigma_\perp)$ by $P.\perp = \mathbf{false}$ and $P.X = (\forall \sigma \in X : P.\sigma)$.

Theorem 2.5 $\qquad wp.S.P = \{\sigma | P.(Op.S.\sigma)\}$

Notice that this means $Op.S_1 = Op.S_2$ if and only if $(\forall P \ : \ wp.S_1.P = wp.S_2.P)$.

[1] $\Sigma_\perp$ denotes the set $\Sigma \cup \{\perp\}$

3 Orderings

In this section we introduce three relations on *Stat*; they are pre-orders, but using Theorem 2.5 they are partial orders when we identify statements with the same operational semantics. We start by two orderings that can be defined by means of weakest preconditions. The first ordering $\sqsubseteq_B$ was proposed by Back [Bac78, Bac80] and is suited for refinement (see [Bac90] and also [Mor87, MRG88]). The second ordering $\sqsubseteq_D$ is a new ordering which preserves deadlocks (as we show below when we give a semantic characterization of the two orderings).

Definition 3.1 *Let $\sqsubseteq_B, \sqsubseteq_D$ be two orderings on Stat defined as follows:*

$$S_1 \sqsubseteq_B S_2 \textit{ if } (\forall Q : wp.S_1.Q \Rightarrow wp.S_2.Q),$$

$$S_1 \sqsubseteq_D S_2 \textit{ if } wp.S_1.\mathbf{false} \Rightarrow wp.S_2.\mathbf{false} \land$$

$$(\forall Q : (wp.S_1.Q \land \neg wp.S_1.\mathbf{false}) \Rightarrow (wp.S_2.Q \land \neg wp.S_2.\mathbf{false})).$$

For the third ordering we need the additional notion of weakest liberal precondition.

Definition 3.2 *(weakest liberal preconditions) Let $wlp : Stat \rightarrow (Pred \rightarrow Pred)$ be defined by*

$$\begin{array}{ll} wlp.b \rightarrow .Q = b \Rightarrow Q & wlp.S_1; S_2.Q = wlp.S_1.(wlp.S_2.Q) \\ wlp.v := t.Q = Q[t/v] & wlp.S_1 \Box S_2.Q = wlp.S_1.Q \land wlp.S_2.Q \\ wlp.loop.Q = \mathbf{true} & wlp.S_1 \Diamond S_2.Q = wlp.S_1.Q \land (wp.S_1.\mathbf{false} \Rightarrow wlp.S_2.Q). \end{array}$$

Note that the weakest liberal precondition differs from the weakest precondition only in the definition of $wlp.loop$ and $wlp.S_1 \Diamond S_2$. The next lemma relates wp and wlp:

Lemma 3.3 $(\forall S, Q : wp.S.Q \Leftrightarrow (wp.S.\mathbf{true} \land wlp.S.Q))$.

Since wp is monotone with respect to the $\Rightarrow$ order, we have by the precedent lemma $(\forall S, Q : wp.S.Q \Rightarrow wlp.S.Q)$. We give a third ordering which was introduced by Nelson in [Nel87].

Definition 3.4 $\quad S_1 \sqsubseteq_N S_2$ *if* $(\forall Q : wp.S_1.Q \Rightarrow wp.S_2.Q \land wlp.S_2.Q \Rightarrow wlp.S_1.Q)$.

The three orderings can be related as follows:

Theorem 3.5 $\quad \sqsubseteq_N \overset{\subset}{\neq} \sqsubseteq_D \overset{\subset}{\neq} \sqsubseteq_B$

Proof We only show the inequalities. They follow from

$$\begin{array}{lll} v := 1 \sqsubseteq_B (\mathbf{false} \rightarrow) & \text{but} & v := 1 \not\sqsubseteq_D (\mathbf{false} \rightarrow) \\ (v := 1 \Box v := 2) \sqsubseteq_D v := 2 & \text{but} & (v := 1 \Box v := 2) \not\sqsubseteq_N v := 2. \end{array}$$

□

We have the following problems with monotonicity of the orderings $\sqsubseteq_B$ and $\sqsubseteq_D$:

1. $$(\mathbf{true} \rightarrow) \sqsubseteq_B (\mathbf{false} \rightarrow)$$
   but
   $$(\mathbf{true} \rightarrow) \Diamond v := 1 \not\sqsubseteq_B (\mathbf{false} \rightarrow) \Diamond v := 1$$

2. $$(v := 1 \Box v := 2) \sqsubseteq_D v := 2$$
   but
   $$(v := 1 \Box v := 2);\ (v = 1 \rightarrow) \not\sqsubseteq_D v := 2;\ (v = 1 \rightarrow).$$

Theorem 3.6 *We have for all statements* $S_1, S_2, S_1', S_2' \in Stat$*:*

$$S_1 \sqsubseteq_B S_2 \wedge S_1' \sqsubseteq_B S_2' \Rightarrow (\forall op \in \{;\,,\Box\} : S_1 op S_1' \sqsubseteq_B S_2 op S_2')$$

$$S_1 \sqsubseteq_D S_2 \wedge S_1' \sqsubseteq_D S_2' \Rightarrow (\forall op \in \{\Box, \Diamond\} : S_1 op S_1' \sqsubseteq_D S_2 op S_2')$$

$$S_1 \sqsubseteq_N S_2 \wedge S_1' \sqsubseteq_N S_2' \Rightarrow (\forall op \in \{;\,,\Box, \Diamond\} : S_1 op S_1' \sqsubseteq_N S_2 op S_2')$$

Proof For $\sqsubseteq_N$ we refer to [Nel87], for $\sqsubseteq_D$ to [BK92] and for $\sqsubseteq_B$ to [BvW90]. □

4 Order Theory

In this section we provide the mathematical basis for the next section. We give some general results on fixed points and we show that under particular conditions they can be obtained (even by iteration) also for non-monotonic functions. Moreover, we give relationships between discrete powerdomains and predicate transformers, following the ideas of [Wan77],[Plo79], [Bes83], [AP86] and [Smy83].

Let P a partial order and A a nonempty subset of P. Then A is said to be *directed* if every finite subset of A has an upper bound. P is a *complete partial order* (cpo) if there exist a least element $\perp$ and every directed subset A of P has least upper bound (lub) $\bigsqcup A$.

For example, for any set X, the *flat* complete partial order $X_\perp$ is the set $X \cup \{\perp\}$ ordered by $x \sqsubseteq y \Leftrightarrow x = \perp$ or $x = y$.

Let P, Q be two partial orders. A function $f : P \rightarrow Q$ is *monotone* if for all $x, y \in P$ with $x \sqsubseteq_P y$ we have $f.x \sqsubseteq_Q f.y$. Moreover, f is *continuous* if for each directed subset A of P with least upper bound $\bigsqcup A$ we have $f.(\bigsqcup A) = \bigsqcup(f.A)$; f is *strict* if and only if

$f.\perp_P = \perp_Q$. If f is continuous then it is monotone, and if f is onto and monotone then it is also strict. Let $g : P \to P$, we denote by $\mu.g$ the least fixed point of g, that is, $g.\mu.g = \mu.g$ and for every other $x \in P$ such that $g.x = x$ then $\mu.g \sqsubseteq x$.

Let P, Q be two partial orders. Then $P \times Q$ is the cartesian product ordered coordinatewise and $P \to Q$ is the function space ordered pointwise. Moreover, if $f^{-1}.y$ exist for $y \in Q$ and $f : P \to Q$ then the partial order determined by $f^{-1}.y$ is the partial order that has for elements $x \in f^{-1}.y \subseteq P$ ordered as in P, that is, for each $x_1, x_2 \in f^{-1}.y$, $x_1 \sqsubseteq x_2 \Leftrightarrow x_1 \sqsubseteq_P x_2$.

4.1 Fixed Points

For any partial order P, function $f : P \to P$ and ordinal λ, define $f^{<\lambda>} \in P$ by

$$f^{<\lambda>} = f.\bigsqcup_{k<\lambda} f^{<k>}.$$

Of course $f^{<\lambda>}$ need not to exist, since $\bigsqcup_{k<\lambda} f^{<k>}$ need not to exist. Note that $f^{<0>} = f.\perp$ when the least element $\perp$ of P exists. If $f^{<\lambda>}$ does not exist , then for any $\lambda' \geq \lambda$ $f^{<\lambda'>}$ does not exist, and if f is monotone then $f^{<\lambda>}$ is monotone in λ. We say $(f^{<\lambda>})_\lambda$ stabilizes at k if whenever $\lambda \geq k$ then $f^{<\lambda>} = f^{<k>}$; the closure ordinal is the least ordinal k by which the sequence stabilizes. If f is monotone then $f^{<k>}$ is the least (pre-)fixed point of f since $f.f^{<k>} = f^{<k+1>}$ and $f.a \sqsubseteq a$ implies $f^{<\lambda>} \sqsubseteq a$ for all λ. If P is a complete partial order and f is monotone then of course $f^{<\lambda>}$ always exists and moreover, $(f^{<\lambda>})_\lambda$ stabilizes [HP72]. If additionally f is continuous then it has closure ordinal $\leq \omega$.

The following theorem, that can be found in [AP86], shows that under certain circumstances $g^{<\lambda>}$ always exists and stabilizes for a monotone function $g : Q \to Q$ even if Q is not a complete partial order:

Theorem 4.1 *Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be two partial orders, and $f : P \to P$, $g : Q \to Q$ be two monotone functions and $h : P \to Q$ be a strict and continuous function such that the following diagram commutes:*

$$\begin{tikzcd}
P \arrow[r, "f"] \arrow[d, "h"'] \arrow[dr, phantom, "*"] & P \arrow[d, "h"] \\
Q \arrow[r, "g"'] & Q
\end{tikzcd}$$

Then if $f^{<\lambda>}$ exists so does $g^{<\lambda>}$, and indeed $g^{<\lambda>} = h.f^{<\lambda>}$. In particular if $\mu.f$ exists (being an $f^{<\lambda>}$) then so does $\mu.g$ and $\mu.g = h.\mu.f$.

We can even drop the condit.on of g to be monotone provided that h satisfies some extra conditions (in [BK92] even a more general theorem is proved but this is not needed here):

Theorem 4.2 *Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be two partial orders, and $f : P \rightarrow P$ be a monotone function, $g : Q \rightarrow Q$ be a function and $h : P \rightarrow Q$ be an onto and monotone function such that for all $y \in Q$ the partial order $h^{-1}.y$ has a top element and the following diagram commutes:*

$$\begin{tikzcd} P \arrow[r, "f"] \arrow[d, "h"'] \arrow[dr, phantom, "*"] & P \arrow[d, "h"] \\ Q \arrow[r, "g"'] & Q \end{tikzcd}$$

Then if $\mu.f$ exists so does $\mu.g$, and indeed $\mu.g = h.\mu.f$. Moreover, if h is also continuous then for each ordinal λ if $f^{<\lambda>}$ exists so does $g^{<\lambda>}$, and $g^{<\lambda>} = h.f^{<\lambda>}$.

Proof The proof contains part of the proof of the Theorem 4.1 [AP86]: assume $\mu.f$ exists, then $\mu.f = f^{<\alpha>}$ for some ordinal α. We have:

$$h.f^{<\alpha>} = h.f^{<\alpha+1>} = h.f.f^{<\alpha>} = g.h.f^{<\alpha>}.$$

So $h.f^{<\alpha>}$ is a fixed point of g. Now it remains to prove that $h.f^{<\alpha>} = \mu.g$. Let $y \in Q$ such that $g.y = y$ and let $a \in P$ be the top element of the partial order generated by $h^{-1}.y$.

First we prove $f.a \sqsubseteq a$, indeed, $f.a \in h^{-1}.y$ because

$$h.f.a = g.h.a = g.y = y$$

and as a is the top element of $h^{-1}.y$ we obtain $f.a \sqsubseteq a$.

As second step we prove by transfinite induction $f^{<\lambda>} \sqsubseteq a$ for each ordinal λ:

$\lambda = 0)$ $f^{<0>} = \perp \sqsubseteq a$

$\lambda > 0)$ { induction hypothesis }

$$(\forall k < \lambda \ : \ f^{<k>} \sqsubseteq a)$$

$\Rightarrow$ { definition of $\bigsqcup$ }

$$\bigsqcup_{k<\lambda} f^{<k>} \sqsubseteq a$$

$\Rightarrow$ { f is monotone }

$$f. \bigsqcup_{k<\lambda} f^{<k>} \sqsubseteq f.a$$

$\Rightarrow$ { definition of $f^{<\lambda>}$ }

$$f^{<\lambda>} \sqsubseteq f.a$$

$\Rightarrow$ { $f.a \sqsubseteq a$ }

$$f^{<\lambda>} \sqsubseteq a$$

Hence also $f^{<\alpha>} \sqsubseteq a$ and by monotonicity of h:

$$h.f^{<\alpha>} \sqsubseteq h.a = y$$

Therefore $h.f^{<\alpha>} = h.\mu.f$ is the least fixed point of g.

Suppose now $f^{<\lambda>}$ exists for some ordinal λ, and let h be continuous. Thus it is also monotone and hence it is also strict as it is onto. The fact that $h.f^{<\lambda>} = g^{<\lambda>}$ follows the line of the proof of the Theorem 4.1 (see [AP86]). □

Note that even if $g : Q \rightarrow Q$ is not monotone and Q is not a complete partial order, the theorem above ensures the existence of a least fixed point for g that can be obtained by iteration, since g^{λ} exists for all ordinals λ.

4.2 Predicate Transformers and Discrete Powerdomains

Let Σ be a nonempty set of states, fixed for the rest of this section, and assume, in order to avoid degenerate cases, its cardinality be greater than 1. Recall that a predicate is a function from states to the boolean set $\mathbf{B} = \{tt, ff\}$. With every predicate $P \in Pred$ we can associate the set $\{\sigma \mid P.\sigma = tt\} \subseteq \Sigma$ while with every set A we can associate the function in *Pred*, $P(A) = \lambda\sigma \in \Sigma.(\text{if } \sigma \in A \text{ then } tt \text{ else } ff)$. If A is a subset of Σ then $A = \{\sigma \mid P(A).\sigma = tt\}$ and conversely, if P is a predicate then $P = P(\{\sigma \mid P.\sigma = tt\})$.

A predicate transformer π is a function in $Pred \rightarrow Pred$ which satisfies some properties. There are different definitions of predicate transformers in the literature that differ in the sets of properties. Next we give a list of possible requirements on the function space $Pred \rightarrow Pred$ that are used in various definitions of predicate transformers:

1. Σ is countable,
2. $\pi.\mathbf{false} = \mathbf{false}$ *(exclusion of miracles)*,
3. π is monotone with respect to the $\Rightarrow$ order,
4. π is continuous with respect to the $\Rightarrow$ order,
5. $\pi.(P \wedge Q) = \pi.P \wedge \pi.Q$ for all $P, Q \in Pred$ *(finite multiplicativity)*,
6. $\pi.\bigwedge_{n \in N} P_n = \bigwedge_{n \in N} \pi.P_n$ where N is the set of natural number and $P_n \in Pred$ for all $n \in N$ *(countable multiplicativity)*,

7. $\pi.\bigwedge_{i\in I} P_i = \bigwedge_{i\in I} \pi.P_i$ where I is an index set of the same cardinality as Σ and $P_i \in \mathit{Pred}$ for all $i \in I$ (*Σ-multiplicativity*),

8. $\pi.\bigwedge_{i\in I} P_i = \bigwedge_{i\in I} \pi.P_i$ where $I \neq \emptyset$ is an index set and $P_i \in \mathit{Pred}$ for all $i \in I$ (*multiplicativity*).

In [Dij76] a predicate transformer $\pi \in \mathit{Pred} \to \mathit{Pred}$ satisfies the properties 1. - 5.; in [Wan77, Plo79] it satisfies the properties 1., 2., 4. and 5.; in [Bes83] the properties 1., 2. and 8.; in [AP86] the properties 1., 2. and 6.; and finally in [BvW90] only the property 3.. A predicate transformer can also satisfy property 7. and we choose this property for defining the predicate transformers that we will use in the rest of the section:

Definition 4.3 *A predicate transformer is any function $\pi \in \mathit{PTran} = \mathit{Pred} \to \mathit{Pred}$ which satisfies the Σ-multiplicativity law.*

Predicate transformers as defined above are stable functions [Plo81], as is shown in the following lemma that is a slight generalization of the stability lemma in [AP86]:

Lemma 4.4 *Let $\pi \in \mathit{PTran}$ and let $\sigma \in \Sigma$ such that $\pi.\mathbf{true}.\sigma$. Then there is a set $min(\pi,\sigma) \subseteq \Sigma$ such that*

$$(\forall Q \;:\; \pi.Q.\sigma \Leftrightarrow min(\pi,\sigma) \subseteq \{\sigma' | Q.\sigma'\}).$$

Next we show some of the relationships among the properties enumerated above:

Lemma 4.5 *Let Σ be a countable set of states. We have:*

$$(4. \;\wedge\; 5.) \Rightarrow 6. \Leftrightarrow 7. \Leftrightarrow 8. \Rightarrow 3.$$

Note that if Σ is uncountable we have 8. $\Leftrightarrow$ 7. $\Rightarrow$ 6. $\Rightarrow$ 3.

The previous lemma shows that predicate transformers as defined in [Dij76] are exactly the same predicate transformers in the sense of [Wan77, Plo79], and these are predicate transformers as defined in [Bes83]. The predicate transformers as defined in [Bes83] are the same predicate transformers defined in [AP86] and these predicate transformers are also predicate transformers in the sense of our definition 4.3. Finally predicate transformers in the sense of our definition 4.3 are also predicate transformers in the sense of [BvW90].

Thus our definition 4.3 generalizes the definitions of [Wan77, Plo79, Bes83, AP86] and we will generalize some of their results. As far as we know similar results do not hold for the definition of predicate transformers of [BvW90]. We will generalize the relationship between the Smyth powerdomain and the predicate transformers [Wan77, Plo79, Bes83, AP86, Smy83] to two our new versions of the Smyth powerdomains. Moreover, we will introduce a relationship between the Egli-Milner powerdomain and pair of predicate transformers like is done in [Nel87]. The following commuting diagram summarizes all the relationships between predicate transformers and discrete powerdomains that we will define in the next three subsections:

$$\begin{tikzcd}
PTran_N \arrow[r, "\cong", "\eta"'] \arrow[d, "\downarrow_1"'] & ETran^{\emptyset} = (\Sigma \arrow[r] \arrow[d, phantom, "*"] & \mathcal{E}^{\emptyset}.\Sigma_\perp) \arrow[d, "e_\Sigma \cdot _"] \\
PTran_D \arrow[r, "\cong", "\gamma"'] \arrow[d, "id_{PTran}"'] & STran^{\delta} = (\Sigma \arrow[r] \arrow[d, phantom, "*"] & \mathcal{S}^{\delta}.\Sigma_\perp) \arrow[d, "d_\Sigma \cdot _"] \\
PTran_B \arrow[r, "\cong", "\omega"'] & STran^{\emptyset} = (\Sigma \arrow[r] & \mathcal{S}^{\emptyset}.\Sigma_\perp)
\end{tikzcd}$$

Egli-Milner powerdomain with empty set

Definition 4.6 *Let $X_\perp$ be a flat domain. Then the Egli-Milner powerdomain with empty set of $X_\perp$, denoted by $\mathcal{E}^{\emptyset}.X_\perp$, is the partial order with elements all the subset of $X_\perp$ ordered as follows:*

$$A \sqsubseteq B \Leftrightarrow (\perp \notin A \ \wedge \ A = B) \ \vee \ (\perp \in A \ \wedge \ A \backslash \{\perp\} \subseteq B).$$

Note that this differs from the usual definition of the Egli-Milner powerdomain because we add the empty set. It is added by means of a smash product following the ideas of [HP79, MM79, Abr91], in fact we have for all $A \subseteq X_\perp$:

$$(A \sqsubseteq \emptyset \Leftrightarrow A = \{\perp\} \ \vee \ A = \emptyset) \text{ and also } (\emptyset \sqsubseteq A \Leftrightarrow A = \emptyset).$$

The partial order $\mathcal{E}^{\emptyset}.X_\perp$ is also complete, as $\{\perp\}$ is the least element and if $\mathcal{F} \subseteq \mathcal{E}^{\emptyset}.X_\perp$ is a directed family then $\bigsqcup \mathcal{F} = (\bigcup \mathcal{F} \backslash \{\perp\}) \cup \{\perp \mid (\forall A \in \mathcal{F} \ : \perp \in A)\}$.

A meaning of a statement will be a function in the *Egli-Milner State-Transformers*, denoted by $ETran^{\emptyset}$, that is, the complete partial order $\Sigma \rightarrow \mathcal{E}^{\emptyset}.\Sigma_\perp$, ordered pointwise. Elements of $\mathcal{E}^{\emptyset}.\Sigma_\perp$ denote resulting computations. Non-terminating computation are represented by the element $\perp$ in the set of all the possible computations. The empty set is interpreted as a deadlock. The Egli-Milner State-Transformers are in the following relation with the predicate transformers (as noted by [Nel87]):

Definition 4.7 *Define the Nelson's predicate transformers $PTran_N$ to be the set of all the functions $\pi \in Pred \rightarrow Pred \times Pred$ such that:*

1. $\downarrow_1 .\pi \in PTran$ *and* $\downarrow_2 .\pi \in PTran$
2. $(\forall Q \in Pred : \downarrow_1 .\pi.\mathbf{true} \wedge \downarrow_2 .\pi.Q \Leftrightarrow \downarrow_1 .\pi_1.Q)$
3. $\downarrow_2 .\pi.\mathbf{true} = \mathbf{true}$

where $\downarrow_i$ denotes a projection operator on the i-th component of the codomain of a function. The functions are ordered as follows

$$\pi \sqsubseteq_{PN} \hat{\pi} \text{ if } (\forall Q \ : \downarrow_1 .\pi.Q \Rightarrow \downarrow_1 .\hat{\pi}.Q \ \wedge \ \downarrow_2 .\hat{\pi}.Q \Rightarrow \downarrow_2 .\pi.Q).$$

By definition of Nelson's predicate transformers we have that $\downarrow_1$: $PTran_N \rightarrow PTran$ is onto, since for each $\pi_1 \in PTran$ the function $\pi : Pred \rightarrow Pred \times Pred$ defined by $\pi.Q = (\pi_1.Q, \pi_2.Q)$ is in $PTran_N$, where

$$\pi_2.Q = \begin{cases} \textbf{true} & \text{if } Q = \textbf{true} \\ \pi_1.Q & \text{otherwise} \end{cases}$$

for all $Q \in Pred$.

For any statement S the pair $(wp.S, wlp.S)$ defined in the definitions 2.4 and 3.2 is a Nelson's predicate transformer and the order $\sqsubseteq_{PN}$ is the lifting of $\sqsubseteq_N$ to $PTran_N$.

Now we can show the relationship between the Egli-Milner powerdomain and the Nelson Predicate Transformers: define the function $\eta : ETran^{\emptyset} \rightarrow PTran_N$, for $m \in ETran^{\emptyset}$ and $P \in Pred$, by

$$\eta.m.P = (\{\sigma | P.m.\sigma\}, \{\sigma | P.(m.\sigma \backslash \{\bot\})\}).$$

Lemma 4.8 *Let $m \in ETran^{\emptyset}$. Then the function $\eta.m \in PTran_N$.*

Lemma 4.9 *The function η is monotone.*

The function η has an inverse. Define the function $\eta^{-1} : PTran_N \rightarrow ETran^{\emptyset}$, for $\pi \in PTran_N$ and $\sigma \in \Sigma$, by:

$$\eta^{-1}.\pi.\sigma = \begin{cases} min(\downarrow_2 .\pi, \sigma) & \text{if } \downarrow_1 .\pi.\textbf{true}.\sigma \\ min(\downarrow_2 .\pi, \sigma) \cup \{\bot\} & \text{otherwise.} \end{cases}$$

Lemma 4.10 *The function η^{-1} is monotone.*

Finally we have:

Theorem 4.11 *The function $\eta : ETran^{\emptyset} \rightarrow PTran_N$ is an isomorphism of partial orders with inverse η^{-1}.*

Smyth powerdomain with deadlock

Definition 4.12 *Let $X_\bot$ be a flat domain. Then the Smyth's powerdomain with deadlock of $X_\bot$, is defined as the partial order*

$$\mathcal{S}^{\delta}.X_\bot = \{A | \ A \subseteq X \ \wedge \ A \neq \emptyset\} \cup \{X_\bot\} \cup \{\delta\}$$

where $A \sqsubseteq B \ \Leftrightarrow \ (A = X_\bot) \vee (A = \delta \ \wedge \ B = \delta) \vee (A \supseteq B)$.

This definition differs from the original definition of the Smyth powerdomain [Smy78] because we add an extra element δ (interpreted as deadlock) that is comparable only with itself and the bottom. This makes that in general $\mathcal{S}^\delta.X_\perp$ is not a complete partial order, in fact consider in $\mathcal{S}^\delta.N_\perp$ the following directed set which has no upper bound:

$$N \sqsubseteq N\backslash\{0\} \sqsubseteq N\backslash\{0,1\} \sqsubseteq ..., \qquad \text{(this example appears also in [AP86]).}$$

The Egli-Milner powerdomain with empty set and Smith powerdomain with deadlock are related by the function $e_X : \mathcal{E}^\emptyset.X_\perp \to \mathcal{S}^\delta.X_\perp$ defined by

$$e_X.A = \begin{cases} A & \text{if } \perp\notin A \ \wedge \ A \neq \emptyset \\ \delta & \text{if } A = \emptyset \\ X_\perp & \text{otherwise} \end{cases}$$

as it is shown in the following lemma:

Lemma 4.13 *The function $e_X : \mathcal{E}^\emptyset.X_\perp \to \mathcal{S}^\delta.X_\perp$ is onto, continuous, and for each $B \in \mathcal{S}^\delta.X_\perp$ the partial order $e_X^{-1}.B$ has a top element.*

We will use this lemma in the next section in order to apply theorem 4.2.

The *Smyth State-Transformers respecting deadlock*, are all the functions $\Sigma \to \mathcal{S}^\delta.\Sigma_\perp$, ordered pointwise. We denote this partial order $STran^\delta$. Elements of $\mathcal{S}^\delta.\Sigma_\perp$ denote resulting computations. All the computations that are possibly non terminating are identified with the element $\{\Sigma_\perp\}$.

Next we show how $STran^\delta$ is related to the predicate transformers. Take $Ptran_D$ as the set of predicate transformers $PTran$ ordered as follows

$$\begin{aligned} \pi \sqsubseteq_{PD} \hat{\pi} \ \text{ if } \ & \pi.\mathbf{false} \Rightarrow \hat{\pi}.\mathbf{false} \ \wedge \\ & \wedge \ (\forall Q \ : \ (\pi.Q \wedge \neg\pi.\mathbf{false}) \Rightarrow \ (\hat{\pi}.Q \wedge \neg\hat{\pi}.\mathbf{false})). \end{aligned}$$

The order $\sqsubseteq_{PD}$ is the lifting of $\sqsubseteq_D$ to $PTran$.

Define for $m \in STran^\delta$ and $Q \in Pred$ the function $\gamma : STran^\delta \to PTran_D$ by

$$\gamma.m.Q = \{\sigma | Q.m.\sigma\} \cup \{\sigma | m.\sigma = \delta\}$$

Define for $\pi \in PTran_D$ and $\sigma \in \Sigma$ the function $\gamma^{-1} : PTran_D \to STran^\delta$ by:

$$\gamma^{-1}.\pi.\sigma = \begin{cases} min(\pi,\sigma) & \text{if } \pi.\mathbf{true}.\sigma \wedge \neg\pi.\mathbf{false}.\sigma \\ \delta & \text{if } \pi.\mathbf{false}.\sigma \\ \Sigma_\perp & \text{otherwise.} \end{cases}$$

Also in this case we have an order-isomorphism:

Theorem 4.14 *The function $\gamma : STran^\delta \to PTran_D$ is an isomorphism of partial orders with inverse γ^{-1}.*

Smyth powerdomain with empty set

Definition 4.15 *Let $X_\perp$ be a flat domain. Then the Smyth powerdomain of $X_\perp$ (with empty set), is defined as the partial order*

$$\mathcal{S}^{\emptyset}.X_\perp = \{A \mid A \subseteq X\} \cup \{X_\perp\}$$

ordered by the superset order, that is, $A \sqsubseteq B \;\Leftrightarrow\; A \supseteq B$.

This definition differs from the original definition of the Smyth's powerdomain [Smy78] because we add the empty set as a top element, as suggested in [Plo79].

The partial order $\mathcal{S}^{\emptyset}.X_\perp$ is also complete, $\{X_\perp\}$ is the least element and if $\mathcal{F} \subseteq \mathcal{S}^{\emptyset}.X_\perp$ is a directed family then $\bigcap \mathcal{F}$ is its least upper bound. Moreover, it is also closed under arbitrary union and intersection.

The *Smyth State-Transformers domain*, denoted by $STran^{\emptyset}$, is the complete partial order $\Sigma \to \mathcal{S}^{\emptyset}.\Sigma_\perp$, ordered pointwise. Elements of $\mathcal{S}^{\emptyset}.\Sigma_\perp$ denote resulting computations. All the computations that are possibly non terminating are identified with the element $\{\Sigma_\perp\}$; the empty set is interpreted as a deadlock.

Also the Egli-Milner powerdomain with empty set and Smith powerdomain with empty set are related by the function $d_X : \mathcal{E}^{\emptyset}.X_\perp \to \mathcal{S}^{\emptyset}.X_\perp$ defined by

$$d_X.A = \begin{cases} A & \text{if } \perp \notin A \\ X_\perp & \text{otherwise} \end{cases}$$

as is shown in the following lemma:

Lemma 4.16 *The function $d_X : \mathcal{E}^{\emptyset}.X_\perp \to \mathcal{S}^{\emptyset}.X_\perp$ is onto, continuous, and for each $B \in \mathcal{S}^{\emptyset}.X_\perp$ the partial order $d_X^{-1}.B$ has a top element.*

We will also use this lemma in the next section in order to apply theorem 4.2.

Next we show the relationship between Smyth state transformers and predicate transformers. Take $PTran_B$ to be the set of predicate transformers $PTran$ ordered pointwise as follows

$$\pi \sqsubseteq_{PB} \hat{\pi} \text{ if } (\forall Q \;:\; \pi.Q \Rightarrow \hat{\pi}.Q).$$

Note that the order $\sqsubseteq_{PB}$ is just the lifting of $\sqsubseteq_B$ to $PTran$.

Define for $m \in STran^{\emptyset}$ and $Q \in Pred$ the function $\omega : STran^{\emptyset} \to PTran_B$ by

$$\omega.m.Q = \{\sigma \mid Q.m.\sigma\}$$

If $m.\sigma = \Sigma_\perp$ then $\omega.m.Q.\sigma = ff$ for all the predicate Q, because $Q.\perp = ff$.

Define for $\pi \in PTran_B$ and $\sigma \in \Sigma$ the function $\omega^{-1} : PTran_B \to STran^{\emptyset}$ by:

$$\omega^{-1}.\pi.\sigma = \begin{cases} min(\pi, \sigma) & \text{if } \pi.\mathbf{true}.\sigma \\ \Sigma_\perp & \text{otherwise.} \end{cases}$$

It is the inverse of ω, indeed we have:

Theorem 4.17 *The function $\omega : STran^{\emptyset} \to PTran_B$ is an isomorphism of partial orders with inverse ω^{-1}.*

5 Recursion

In this section we add recursion to the language. Let $(x \in)PVar$ be a nonempty set of procedure variables. We remove *loop* from and add procedure variables to the set of statements *Stat*: it is now given by

$$S ::= x \mid v := t \mid b \to \mid S_1; S_2 \mid S_1 \Box S_2 \mid S_1 \Diamond S_2.$$

For the semantics we introduce the set of environments $Env = (PVar \to PTran)$, that is, an environment gives a predicate transformer for each procedure variable.

Next we give the extension of *wp* and *wlp* as defined in definition 2.4 and definition 3.2 to the new set of statements:

Definition 5.1 *(Extension of wp and wlp) Let $wp : Stat \to (Env \to PTran)$ for $\xi \in Env$ be defined by*

$$\begin{array}{ll} wp.b \to .\xi.Q = b \Rightarrow Q & wp.S_1; S_2.\xi.Q = wp.S_1.\xi.(wp.S_2.\xi.Q) \\ wp.x.\xi.Q = \xi.x.Q & wp.S_1 \Box S_2.\xi.Q = wp.S_1.\xi.Q \wedge wp.S_2.\xi.Q \\ wp.v := t.\xi.Q = Q[t/v] & wp.S_1 \Diamond S_2.\xi.Q = wp.S_1.\xi.Q \wedge (wp.S_1.\xi.\mathbf{false} \Rightarrow wp.S_2.\xi.Q) \end{array}$$

and let $wlp : Stat \to (Env \to PTran)$ be extended in similar way.

Take a fixed declaration $d \in Decl : Pvar \to Stat$. Sometimes we denote $d.x = S$ by $x \Leftarrow S$. A declaration assigns to each procedure variable a statement, possibly containing procedure variables. The idea is to associate with a declaration an environment by means of a fixed point construction.

First we show how familiar constructions can be defined in a declaration: the do-loop **do** S **od** can be defined by $x \Leftarrow (S; x) \Diamond (\mathbf{true} \to)$ and *loop* by $x \Leftarrow x$.

Define $\phi : Decl \to (Env \to Env)$ for $\xi \in Env$ by

$$\phi.d.\xi.x = wp.(d.x).\xi.$$

We would like to show that $(\phi.d)$ has a (least) fixed point (for any declaration d) that can be obtained by iteration, such that we can take this fixed point as the meaning of the declaration.

In order to do this we lift *Env* to the partial orders $(Env_B, \sqsubseteq_{EB}), (Env_D, \sqsubseteq_{ED})$ and $(Env_N, \sqsubseteq_{EN})$ defined, respectively, by

- $Env_B = (PVar \to PTran_B)$ and $\xi_1 \sqsubseteq_{EB} \xi_2$ *if* $(\forall x \in PVar : \xi_1.x \sqsubseteq_{PB} \xi_2.x)$
- $Env_D = (PVar \to PTran_D)$ and $\xi_1 \sqsubseteq_{ED} \xi_2$ *if* $(\forall x \in PVar : \xi_1.x \sqsubseteq_{PD} \xi_2.x)$
- $Env_N = (PVar \to PTran_N)$ and $\xi_1 \sqsubseteq_{EN} \xi_2$ *if* $(\forall x \in PVar : \xi_1.x \sqsubseteq_{PN} \xi_2.x)$.

Theorem 5.2 $(Env_N, \sqsubseteq_{EN})$ *is a complete partial ordering.*

Lift the definition above of ϕ to $\phi_k : Decl \to (Env_k \to Env_k)$, for $k \in \{B, D, N\}$ and $\xi_k \in Env_k$, by

$$\phi_k.d.\xi_k.x = \begin{cases} wp.(d.x).\xi_k & \text{if } k \in \{B, D\} \\ (wp.(d.x).\downarrow_1.\xi_k, wlp.(d.x).\downarrow_2.\xi_k) & \text{if } k = N. \end{cases}$$

The main problem is that fcr a fixed declaration d the functions $(\phi_B.d)$ and $(\phi_D.d)$ are in general not monotone (adapt the examples at the end of section 3).

However, define two functions $h_{NB} : Env_N \to Env_B$ and $h_{ND} : Env_N \to Env_D$ by:

$$(\forall \xi \in Env_N \ : \ h_{NB}.\xi = h_{ND}.\xi = \downarrow_1.\xi).$$

Using the results of the previous section, we have that both h_{NB} and h_{ND} are onto, continuous and for every $\xi \in Env_B$ there is a top element in $h_{NB}^{-1}.\xi$, and similarly for every $\xi \in Env_D$ there is a top element in $h_{ND}^{-1}.\xi$.

Hence we can apply the theorem 4.2:

Theorem 5.3 *The function $(\phi_k.d)$ defined above has for a fixed declaration d a least fixed point $\mu.(\phi_k.d)$ both with respect to $\sqsubseteq_{EB}$, $\sqsubseteq_{ED}$ and $\sqsubseteq_{EN}$ that can be obtained by iteration as follows: define $\xi^{<0>}$ the environment such that for all x and Q*

$$\xi^{<0>}.x.Q = \textbf{false}$$

and define for each ordinal $\lambda > 0$

$$\xi^{<\lambda>} = \phi_k.d.\bigsqcup_{\alpha<\lambda} \xi^{<\alpha>},$$

then there is an ordinal $\hat{\lambda}$ such that $\mu.(\phi_k.d) = \xi^{<\hat{\lambda}>}$.

Finally we can give the following three weakest precondition semantics:

Definition 5.4 *Let $S \in Stat$, $d \in Decl$ and $k \in \{B, D, N\}$. We define the following three weakest precondition semantics $\mathcal{W}_k : Stat \to (Decl \to PTran_k)$ by:*

- $\mathcal{W}_B.S.d = wp.S.(\mu.(\phi_B.d))$
- $\mathcal{W}_D.S.d = wp.S.(\mu.(\phi_D.d))$
- $\mathcal{W}_N.S.d = (wp.S.\downarrow_1.(\mu.(\phi_N.d)), wlp.S.\downarrow_2.(\mu.(\phi_N.d)))$.

Acknowledgements

We like to acknowledge all the members of the Amsterdam Concurrency Group especially Jaco de Bakker, Franck van Breugel, Jan Rutten, and Daniele Turi, for discussions and suggestions about the contents of this paper. Thanks also to Ralph Back, Manfred Broy, Lambert Meertens, Doaitse Swierstra, Kaisa Sere, Rob Udink, Hans Zantema, and Prakash Panaganden.

References

[Abr91] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.

[AP86] K. R. Apt and G. Plotkin. Countable nondeterminism and random assignment. *Journal of the ACM*, 33(4):724–767, October 1986.

[Bac78] R.-J.R. Back. *On the correctness of Refinement Steps in Program Development.* PhD thesis, Department of Computer Science, University of Helsinki, 1978. Report A-1978-4.

[Bac80] R.-J.R. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications*, volume 131 of *Mathematical Centre Tracts.* Mathematical Centre, Amsterdam, 1980.

[Bac90] R.-J.R. Back. Refinement calculus, part ii: Parallel and reactive programs. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, number 430 in Lecture Notes in Computer Science, pages 67–93, 1990.

[Bak80] J. W. de Bakker. *Mathematical Theory of Program Corretness.* Prentice-Hall, 1980.

[Bes83] E. Best. Relational semantic of concurrent programs (with some applications). In D. Bjorner, editor, *Proc. of the IFIP Working Conference on on Formal Description of Programming Concepts - II*, pages 431–452, Garmisch-Partenkirchen, FRG, 1983. North-Holland Publishing Company.

[BK92] M. M. Bonsangue and J. N. Kok. Semantics, orderings and recursion in the weakest precondition calculus. Technical report, Centre for Mathematics and Computer Science, Amsterdam, 1992. To appear.

[BvW90] R.-J.R. Back and J. von Wright. Refinement calculus, part i: Sequential nondeterministic programs. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, number 430 in Lecture Notes in Computer Science, pages 42–66, 1990.

[Dij76] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[Hes89] W.H. Hesselink. Predicate transformer semantics of general recursion. *Acta Informatica*, 26:309–332, 1989.

[HP72] P. Hitchcock and D. Park. Induction rules and termination proofs. In *International Conference on Automata, Languages and Programming*, 1972.

[HP79] M. Hennessy and G. D. Plotkin. Full abstraction for a simple parallel programming language. In J. Becvar, editor, *Proc. 8th Int'l Symp. on Mathematical Foundations on Computer Science*, volume 74 of *Lecture Notes in Computer Science*, pages 108–120. Springer-Verlag, Berlin, 1979.

[MM79] G. Milne and R. Milner. Concurrent processes and their syntax. *J. ACM*, 26, 2:302–321, 1979.

[Mor87] J. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9:287–306, 1987.

[MRG88] C.C. Morgan, K.A. Robinson, and P.H.B. Gardiner. On the refinement calculus. Technical Report PRG–70, Programming Research Group, 1988.

[Nel87] G. Nelson. A generalization of Dijkstra's calculus. Technical Report 16, Digital Systems Research Center, 1987.

[Plo79] G. D. Plotkin. Dijkstra's predicate transformer and Smyth's powerdomain. In *Proceedings of the Winter School on Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 527–553. Springer-Verlag, Berlin, 1979.

[Plo81] G.D. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). Department of Computer Science, Univ. of Edinburgh, 1981.

[Smy78] M.B. Smyth. Power domains. *J. Comput. Syst. Sci.*, 16,1:23–36, 1978.

[Smy83] M.B. Smyth. Power domains and predicate transformers: A topological view. In *Proceeding of ICALP '83 (Barcelona)*, volume 154 of *Lecture Notes in Computer Science*, pages 662–675. Springer-Verlag, Berlin, 1983.

[Wan77] M. Wand. A characterisation of weakest preconditions. *J. Comput. Syst. Sci.*, 15:209–212, 1977.