Semantics, Orderings and Recursion in the Weakest Precondition Calculus

Marcello Bonsangue and Joost N. Kok

RUU-CS-92-40
December 1992

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Semantics, Orderings and Recursion in the Weakest Precondition Calculus

Marcello Bonsangue and Joost N. Kok

Technical Report RUU-CS-92-40
December 1992

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0024-3275

Semantics, Orderings and Recursion in the Weakest Precondition Calculus

Marcello Bonsangue
Centrum voor Wiskunde en Informatica
P. O. Box 4079, 1009 AB Amsterdam, The Netherlands.
marcello@cwi.nl.

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

Abstract: An extension of Dijkstra's guarded command language is studied, including sequential composition, demonic choice and a backtrack operator. 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 theory of fixed points in partial orderings 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. In order to give a semantic characterization of the three orderings that relates initial states to possible outcomes of the computation, the relations between predicate transformers and discrete powerdomains is studied. Three powerdomains are considered: two versions of the Smyth powerdomain and the Egli-Milner powerdomain. For each of them an isomorphism is proved with a suitable domain of predicate transformers.

1991 Mathematics Subject Classification: 68Q55, 68Q10, 68Q60, 06A06.

1991 CR Categories: D.3.1, F.1.2, F.3.1, F.3.2.

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

Note: The research of Marcello Bonsangue was initially supported by a grant of the Universita' degli Studi di Milano, Italy, and later by a grant of the Centro Nazionale delle Ricerche (CNR), Italy.

Contents

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]). This 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.

An early treatment of recursion, depending on continuity, was given in [dR76], and a more detailed treatment was given in [Heh79] and [Bak80]. Inspired by this last work, 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.

A different approach for the addition of recursive procedures is to consider fixed point equations over predicates instead of recursion over predicate transformers, (see a discussion in [Nel87]) but this approach is not considered here.

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). We guide the intuition by giving a compositional semantic model for this language that relates initial states to possible outcomes and prove this model equivalent to the weakest precondition model.

Then we consider three orderings; besides the orderings of Back and Nelson we define a new ordering in between. This ordering was derived from an ordering which we used for refinement of PROLOG programs. 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. Also, Back's ordering is semantically characterized by relating it to the semantic model.

Only Nelson's ordering is monotonic with respect to all three operators, in fact 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 least fixed point. Then Apt and Ploktin [AP81],[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. The use of commutative diagrams to transfer fixed point properties from a domain to another has been explored in detail in Meyer [Mey85]. Finally, here 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. This last result implies 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.

Next we provide a semantic characterization of the three weakest precondition models based on a denotational semantic model for the language that relates initial states to possible outcomes of the computation. We start from the well known duality theory connecting the discrete version of the Smyth powerdomain [Smy78] and the Dijkstra's predicate transformers [Wan77, Plo79, Smy83, Bes83, AP86]: there is an order isomorphism between functions from a set of states to the Smyth powerdomain (ordered pointwise) and the predicate transformers (ordered by a refinement order).

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. Thus, 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 (extended with the empty set too, in in order to treat deadlock) showing an isomorphism between the Egli Milner state transformation and the Nelson's predicate transformers like is done in [Nel87].

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. Composition of functions is denoted by the circle $\circ$, while function application is denoted by . and associates to the left:

$$f.g.x = (f.g).x$$

We use the currying convention that a function with more than one argument is treated as a function of the first argument which yields a function of the remaining arguments.

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 sequential composition execute the first component and then execute the second. The demonic choice executes the first or the second component while 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 $b \rightarrow$. To guide the intuition about this language we give an operational semantic model below that relates initial states with possible outcomes of the computation.

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 as the statement $(b_1 \rightarrow ; S_1 \Box b_2 \rightarrow ; S_2) \Diamond loop$. More general derived statements are **if** S **fi** $= S \Diamond loop$, $skip =$ **true** $\rightarrow$, $abort = loop$, and $magic =$ **false** $\rightarrow$.

Next we turn to the operational semantic model. The set of states $(\sigma \in)\Sigma$ is given by the function space $\Sigma = Var \rightarrow N$.

A state is a function that yields an integer for each variable in Var. 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:*

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

$$\langle b \rightarrow, \sigma\rangle \longrightarrow \langle E, \sigma\rangle \quad \textit{if } b.\sigma$$

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

$$\langle \mathit{loop}, \sigma\rangle \longrightarrow \langle \mathit{loop}, \sigma\rangle$$

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

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

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

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

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

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

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

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 $c_0 = \langle m, \sigma\rangle$. We also say that m cannot diverge in σ (denoted by $(m, \sigma)\downarrow$) if not $(m, \sigma)\uparrow$. By $c \longrightarrow^* c'$ we denote that there exists a finite sequence

$$c \longrightarrow c_1 \cdots c_n \longrightarrow c'.$$

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:*

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

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

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. For example, consider the statement $(x := 1; x = 0 \rightarrow\ ; x := 2) \Diamond x := 3$ and let $\sigma \in \Sigma$ such that $\sigma.x = 0$. Then we have:

$$\frac{\langle x := 1, \sigma\rangle \longrightarrow \langle E, \sigma'\rangle}{\langle (x := 1; x = 0 \rightarrow\ ; x := 2) \Diamond x := 3, \sigma\rangle \longrightarrow \langle (x = 0 \rightarrow\ ; x := 2) \triangle (x := 3, \sigma), \sigma'\rangle},$$

where $\sigma'.x = 1$, and also

$$\frac{\langle (x = 0 \rightarrow\ ; x := 2), \sigma'\rangle \longrightarrow \langle E, \delta\rangle \ \wedge\ \langle x := 3, \sigma\rangle \longrightarrow \langle E, \sigma"\rangle}{\langle (x = 0 \rightarrow\ ; x := 2) \triangle (x := 3, \sigma), \sigma'\rangle \longrightarrow \langle E, \sigma"\rangle},$$

where $\sigma".x = 3$ and $\langle (x = 0 \rightarrow\ ; x := 2), \sigma'\rangle \longrightarrow \langle E, \delta\rangle$ because

$$\frac{\langle x = 0 \rightarrow, \sigma'\rangle \longrightarrow \langle E, \delta\rangle}{\langle (x = 0 \rightarrow\ ; x := 2), \sigma'\rangle \longrightarrow \langle E, \delta\rangle}.$$

Therefore

$$\langle (x := 2; x = 0 \rightarrow\ ; x := 2) \Diamond x := 3, \sigma\rangle \longrightarrow^* \langle E, \sigma"\rangle.$$

To give some more feeling for this transition system we give the following examples of equalities between statements (an equality $S_1 =_{Op} S_2$ between two statements denotes that S_1 and S_2 have the same operational semantics, that is, $Op.S_1 = Op.S_2$): For all $S \in Stat$ we have

$$(\mathbf{false} \rightarrow \Box S) =_{Op} (S \Box \mathbf{false} \rightarrow) =_{Op} S, \quad (loop \Box S) =_{Op} (S \Box loop) =_{Op} loop,$$

$$(\mathbf{false} \rightarrow \Diamond S) =_{Op} S, \qquad (loop \Diamond S) =_{Op} loop,$$

$$(\mathbf{false} \rightarrow\ ; S) =_{Op} \mathbf{false} \rightarrow, \qquad (\mathbf{true} \rightarrow\ ; S) =_{Op} (S; \mathbf{true} \rightarrow) =_{Op} S,$$

$$(loop; S) =_{Op} loop.$$

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. With every predicate we can associate a set $\{\sigma | P.\sigma = tt\}$.

Definition 2.4 *(weakest precondition) Let*

$$wp\ :\ Stat \rightarrow (Pred \rightarrow Pred)$$

be defined as follows:

$$wp.b \rightarrow .Q = b \Rightarrow Q$$

$$wp.v := t.Q = Q[t/v]$$

$$wp.loop.Q = \textbf{false}$$

$$wp.S_1; S_2.Q = wp.S_1.(wp.S_2.Q)$$

$$wp.S_1 \Box S_2.Q = wp.S_1.Q \wedge wp.S_2.Q$$

$$wp.S_1 \Diamond S_2.Q = wp.S_1.Q \wedge (wp.S_1.\textbf{false} \Rightarrow wp.S_2.Q).$$

(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 as in [Bak80]; at first let us generalize predicates P from Σ to $(\mathcal{P}.\Sigma \cup \Sigma_\perp)$ by $P.\perp = \textbf{false}$ and $P.X = (\forall \sigma \in X : P.\sigma)$.

Theorem 2.5 *Let $S \in Stat$ and $P \in Pred$. Then*

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

The proof is given in the appendix. The theorem above gives us the following operational characterizations of some interesting weakest preconditions:

Corollary 2.6 *For all the statement S we have:*

1. $wp.S.\textbf{false} = \{\sigma | Op.S.\sigma = \emptyset\}$,
2. $(\neg wp.S.\textbf{false} \wedge wp.S.\textbf{true}) = \{\sigma | Op.S.\sigma \neq \perp \ \wedge \ Op.S.\sigma \neq \emptyset\}$,
3. $(\neg wp.S.\textbf{true}) = \{\sigma | Op.S.\sigma = \perp\}$.

Furthermore, we have the following relation between the operational semantics and the weakest precondition semantics:

Corollary 2.7 *Let $S_1, S_2 \in Stat$ and $P \in Pred$. Then*

$$Op.S_1 = Op.S_2 \Leftrightarrow (\forall P \ : \ wp.S_1.P = wp.S_2.P).$$

3 Orderings

In this section we introduce three relations on *Stat*; they are pre-orders, but using Corollary 2.7 they are partial orders when we identify statements with the same operational semantic *Op*. 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:*

- $S_1 \sqsubseteq_B S_2$ *if* $(\forall Q : wp.S_1.Q \Rightarrow wp.S_2.Q)$,
- $S_1 \sqsubseteq_D S_2$ *if* $wp.S_1.\mathbf{false} \Rightarrow wp.S_2.\mathbf{false} \wedge$
  $(\forall Q : (wp.S_1.Q \wedge \neg wp.S_1.\mathbf{false}) \Rightarrow (wp.S_2.Q \wedge \neg wp.S_2.\mathbf{false}))$.

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

Definition 3.2 *(weakest liberal precondition) Let*

$$wlp : Stat \rightarrow (Pred \rightarrow Pred)$$

be defined by

$$wlp.b \rightarrow .Q = b \Rightarrow Q$$

$$wlp.v := t.Q = Q[t/v]$$

$$wlp.loop.Q = \mathbf{true}$$

$$wlp.S_1; S_2.Q = wlp.S_1.(wlp.S_2.Q)$$

$$wlp.S_1 \Box S_2.Q = wlp.S_1.Q \wedge wlp.S_2.Q$$

$$wlp.S_1 \Diamond S_2.Q = wlp.S_1.Q \wedge (wp.S_1.\mathbf{false} \Rightarrow wlp.S_2.Q).$$

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 *Let $S \in Stat$ and $Q \in Pred$. Then*

$$wp.S.Q \Leftrightarrow (wp.S.\mathbf{true} \wedge wlp.S.Q).$$

Since wp is monotone we have $wp.S.Q \Rightarrow wp.S.\mathbf{true}$, and hence by the theorem above also $wp.S.Q \Rightarrow wlp.S.Q$. Now we can give a third ordering which was introduced by Nelson in [Nel87].

Definition 3.4 *Let $\sqsubseteq_N$ be the ordering on Stat defined as follows:*

$$S_1 \sqsubseteq_N S_2 \text{ if } (\forall Q : wp.S_1.Q \Rightarrow wp.S_2.Q \wedge wlp.S_2.Q \Rightarrow wlp.S_1.Q).$$

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 The inequalities follow from

- $v := 1 \sqsubseteq_B (\textbf{false} \rightarrow)$ but $v := 1 \not\sqsubseteq_D (\textbf{false} \rightarrow)$,
- $(v := 1 \Box v := 2) \sqsubseteq_D v := 2$ but $(v := 1 \Box v := 2) \not\sqsubseteq_N v := 2$.

($\sqsubseteq_N \subseteq \sqsubseteq_D$) Take two arbitrary statements S_1, S_2 and a predicate Q. Assume that $S_1 \sqsubseteq_N S_2$ and $(wp.S_1.Q \wedge \neg wp.S_1.\textbf{false})$. We prove $(wp.S_2.Q \wedge \neg wp.S_2.\textbf{false})$:

$(wp.S_1.Q \wedge \neg wp.S_1.\textbf{false})$

$=$ { lemma 3.3, $wp.S_1.\textbf{false} = (wp.S_1.\textbf{true} \wedge wlp.S_1.\textbf{false})$ }

$wp.S_1.Q \wedge (\neg wp.S_1.\textbf{true} \vee \neg wlp.S_1.\textbf{false})$

$\Rightarrow$ { Monotonicity $wp.S_1$, $Q \Rightarrow \textbf{true}$, $wp.S_1.Q \Rightarrow wp.S_1.\textbf{true}$ }

$wp.S_1.Q \wedge (\neg wp.S_1.Q \vee \neg wlp.S_1.\textbf{false})$

$=$

$(wp.S_1.Q \wedge \neg wp.S_1.Q) \vee (wp.S_1.Q \wedge \neg wlp.S_1.\textbf{false})$

$=$

$\textbf{false} \vee (wp.S_1.Q \wedge \neg wlp.S_1.\textbf{false})$

$\Rightarrow$ { $(\forall P : \textbf{false} \Rightarrow P)$ }

$(wp.S_1.Q \wedge \neg wlp.S_1.\textbf{false})$

$\Rightarrow \{S_1 \sqsubseteq_N S_2\}$

$wp.S_2.Q. \wedge \neg wlp.S_2.\textbf{false}$

$\Rightarrow \{(\forall S, Q : wp.S.Q \Rightarrow wlp.S.Q)\}$

$wp.S_2.Q. \wedge \neg wp.S_2.\textbf{false}.$

($\sqsubseteq_D \subseteq \sqsubseteq_B$) Take two arbitrary statements S_1, S_2 and a predicate Q. Assume that $S_1 \sqsubseteq_D S_2$ and $wp.S_1.Q$. We are going to prove $wp.S_2.Q$.

$wp.S_1.Q$

$=$

$(wp.S_1.Q \wedge wp.S_1.\textbf{false}) \vee (wp.S_1.Q \wedge \neg wp.S_1.\textbf{false})$

$=$ { Monotonicity $wp.S_1$: $wp.S_1.\textbf{false} \Rightarrow wp.S_1.Q$ }

$wp.S_1.\textbf{false} \vee (wp.S_1.Q \wedge \neg wp.S_1.\textbf{false})$

$\Rightarrow$ { $S_1 \sqsubseteq_D S_2$ }

$$wp.S_2.\mathbf{false} \vee (wp.S_2.Q \wedge \neg wp.S_2.\mathbf{false})$$

$= \{$ Monotonicity $wp.S_2$: $wp.S_2.\mathbf{false} \Rightarrow wp.S_2.Q$ $\}$

$$(wp.S_2.Q \wedge wp.S_2.\mathbf{false}) \vee (wp.S_2.Q \wedge \neg wp.S_2.\mathbf{false})$$

$=$

$$wp.S_2.Q.$$

□

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

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

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

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

1. $S_1 \sqsubseteq_B S_2 \wedge S_1' \sqsubseteq_B S_2' \Rightarrow (\forall op \in \{;, \Box\} : S_1 op S_1' \sqsubseteq_B S_2 op S_2')$,
2. $S_1 \sqsubseteq_D S_2 \wedge S_1' \sqsubseteq_D S_2' \Rightarrow (\forall op \in \{\Box, \Diamond\} : S_1 op S_1' \sqsubseteq_D S_2 op S_2')$,
3. $S_1 \sqsubseteq_N S_2 \wedge S_1' \sqsubseteq_N S_2' \Rightarrow (\forall op \in \{;, \Box, \Diamond\} : S_1 op S_1' \sqsubseteq_N S_2 op S_2')$.

Proof For the orderings $\sqsubseteq_B$ and $\sqsubseteq_N$ we refer to [BvW90] and [Nel87], respectively. It remains to prove the result for $\sqsubseteq_D$:

- The operator $\Box$ is monotone:

  $$wp.S_1 \Box S_1'.\mathbf{false}$$

  $=$ {definition of wp}

  $$wp.S_1.\mathbf{false} \wedge wp.S_1'.\mathbf{false}$$

  $\Rightarrow$ { $S_1 \sqsubseteq_D S_2$ and $S_1' \sqsubseteq_D S_2'$}

  $$wp.S_2.\mathbf{false} \wedge wp.S_2'.\mathbf{false}$$

  $=$

  $$wp.S_2 \Box S_2'.\mathbf{false}.$$

  Moreover, we have:

  $$wp.S_1 \Box S_1'.Q \wedge \neg wp.S_1 \Box S_1'.\mathbf{false}$$

  $=$ {definition of wp}

$$wp.S_1.Q \wedge wp.S_1'.Q \wedge (\neg wp.S_1.\mathbf{false} \vee \neg wp.S_1'.\mathbf{false})$$

$=$

$$(wp.S_1.Q \wedge \neg wp.S_1.\mathbf{false} \wedge wp.S_1'.Q) \vee (wp.S_1.Q \wedge wp.S_1'.Q \wedge \neg wp.S_1'.\mathbf{false})$$

$\Rightarrow$ { $S_1 \sqsubseteq_D S_2$ and $S_1' \sqsubseteq_D S_2'$ and $\sqsubseteq_D \subseteq \sqsubseteq_B$ }

$$(wp.S_2.Q \wedge \neg wp.S_2.\mathbf{false} \wedge wp.S_2'.Q) \vee (wp.S_2.Q \wedge wp.S_2'.Q \wedge \neg wp.S_2'.\mathbf{false})$$

$=$

$$wp.S_2.Q \wedge wp.S_2'.Q \wedge (\neg wp.S_2.\mathbf{false} \vee \neg wp.S_2'.\mathbf{false})$$

$=$

$$wp.S_2 \Box S_2'.Q \wedge \neg wp.S_2 \Box S_2'.\mathbf{false}.$$

- The operator $\Diamond$ is monotone:

  $$wp.S_1 \Diamond S_1'.\mathbf{false}$$

  $=$ {definition of wp and $\Rightarrow$}

  $$wp.S_1.\mathbf{false} \wedge ((wp.S_1.\mathbf{false} \wedge wp.S_1'.\mathbf{false}) \vee \neg wp.S_1.\mathbf{false})$$

  $=$ { calculation }

  $$wp.S_1.\mathbf{false} \wedge wp.S_1'.\mathbf{false}$$

  $\Rightarrow$ { $S_1 \sqsubseteq_D S_2$ and $S_1' \sqsubseteq_D S_2'$ }

  $$wp.S_2.\mathbf{false} \wedge wp.S_2'.\mathbf{false}$$

  $=$

  $$wp.S_2.\mathbf{false} \wedge ((wp.S_2.\mathbf{false} \wedge wp.S_2'.\mathbf{false}) \vee \neg wp.S_2.\mathbf{false})$$

  $=$

  $$wp.S_2 \Diamond S_2'.\mathbf{false}.$$

  Moreover, we have:

  $$wp.S_1 \Diamond S_1'.Q \wedge \neg wp.S_1 \Diamond S_1'.\mathbf{false}$$

  $=$ {definition of wp and $\Rightarrow$}

  $$wp.S_1.Q \wedge ((wp.S_1.\mathbf{false} \wedge wp.S_1'.Q) \vee \neg wp.S_1.\mathbf{false}) \wedge$$

  $$(\neg wp.S_1.\mathbf{false} \vee (wp.S_1.\mathbf{false} \wedge \neg wp.S_1'.\mathbf{false}))$$

$= \{$ calculation $\}$

$$(wp.S_1.Q \wedge \neg wp.S_1.\mathbf{false}) \vee (wp.S_1.Q \wedge wp.S_1.\mathbf{false} \wedge wp.S_1'.Q \wedge \neg wp.S_1'.\mathbf{false})$$

$\Rightarrow \{ S_1 \sqsubseteq_D S_2$ and $S_1' \sqsubseteq_D S_2'$ and $\sqsubseteq_D \subseteq \sqsubseteq_B \}$

$$(wp.S_2.Q \wedge \neg wp.S_2.\mathbf{false}) \vee (wp.S_2.Q \wedge wp.S_2.\mathbf{false} \wedge wp.S_2'.Q \wedge \neg wp.S_2'.\mathbf{false})$$

$=$

$$wp.S_2.Q \wedge ((wp.S_2.\mathbf{false} \wedge wp.S_2'.Q) \vee \neg wp.S_2.\mathbf{false}) \wedge$$

$$(\neg wp.S_2.\mathbf{false} \vee (wp.S_2.\mathbf{false} \wedge \neg wp.S_2'.\mathbf{false}))$$

$=$

$$wp.S_2 \diamond S_2'.Q \wedge \neg wp.S_2 \diamond S_2'.\mathbf{false}.$$

□

If we do not allow $\Box$ as an operator in the set of statements then all statements S are deterministic (that is, $Op.S.\sigma \in \Sigma \cup \{\bot\}$ or $Op.S.\sigma = \emptyset$ for all states σ). For this deterministic subset of *Stat* the ordering $\sqsubseteq_D$ is monotone. This subset with sequential composition and the backtrack operator has a close correspondence with the language PROLOG. The ordering $\sqsubseteq_D$ can be used as a refinement relation for PROLOG programs and we hope to return to this connection in future work.

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 some relationships between discrete powerdomains and predicate transformers, generalizing ideas of [Wan77],[Plo79], [Bes83], [Smy83], and[AP86]. Most of the standard notions on domain Theory can be found, for example, in [Plo81].

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 $\bot$ and every directed subset A of P has least upper bound (lub) $\bigsqcup A$. Moreover, we have that A is an *antichain* of P if and only if $(\forall a, b \in A \ : \ a \sqsubseteq b \vee b \sqsubseteq a \ \Rightarrow a = b)$; an antichain A is an *upper fringe* of P if and only if $(\forall x \in P \backslash A \ \forall a \in A \ : \ x \sqsubseteq a)$; and an antichain A is a *lower fringe* of P if and only if $(\forall x \in P \backslash A \ \forall a \in A \ : \ a \sqsubseteq x)$.

For example, for any set X, the *flat* complete partial order $X_\bot$ is the set $X \cup \{\bot\}$ ordered by $x \sqsubseteq y \Leftrightarrow x = \bot$ or $x = y$. Then all the subsets of X, and $\{\bot\}$ are antichains, the set X is the only upper fringe and $\{\bot\}$ is the only lower fringe.

Lemma 4.1 *Let P be a partial order. If A is an upper (lower) fringe of P then it is unique.*

Proof Assume B is another upper (lower) fringe of P different from A. Being A and B upper (lower) fringes they are non-empty by definition, thus there are only two possibilities:

i) there is an element $b \in B$ but $b \notin A$.
Take an $a \in A$. We have $b \sqsubseteq a (a \sqsubseteq b)$ because A is an upper (lower) fringe. But also B is an upper (lower) fringe thus either $a \in B$ and hence $a = b$ because an upper fringe is an antichain, or $a \notin B$ and hence $a \sqsubseteq b$ that implies $a = b$. But this contradicts $b \notin A$.

ii) there is an element $a \in A$ but $a \notin B$.
As above.

□

Note that if the partial order P is finite then every antichain of P is finite too. Moreover, if P has a top element x then clearly $\{x\}$ is the finite upper fringe of P, and similarly if P has a bottom element y then $\{y\}$ is the finite lower fringe of P.

Let P, Q be two partial orders. A function $f : P \to Q$ 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 $f.\ \bot_P = \bot_Q$. 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 $\mu.g \sqsubseteq x$.

Lemma 4.2 *Let P, Q be two partial orders and $f : P \to Q$ be a continuous function. Then f is monotone.*

Proof Let $x \sqsubseteq_P y$. Then the set $\{x, y\}$ is directed as every finite subset has upper bound. Moreover, $\bigsqcup\{x, y\} = y$, indeed, $x \sqsubseteq_P y$ and $y \sqsubseteq_P y$, and for each other $z \in P$ such that $x \sqsubseteq_P z$ and $y \sqsubseteq_P z$ then $\bigsqcup\{x, y\} = y \sqsubseteq_P z$. Hence $f.y = f.\bigsqcup\{x, y\} = \bigsqcup\{f.x, f.y\}$ as f is continuous and this implies $f.x \sqsubseteq_Q f.y$. □

Upper fringe, lower fringe and antichains have the following property:

Lemma 4.3 *Let P be a partial order and $f : P \to P$ be a monotone function.*

1. *If P has finite upper fringe A then there exist an $a \in A$ and a natural number $n > 0$ such that $f^n.a \sqsubseteq a$,*
2. *If P has finite lower fringe A then there exist an $a \in A$ and a natural number $n > 0$ such that $a \sqsubseteq f^n.a$,*
3. *If every antichain of P is finite then there exist an $x \in P$ and a natural number $n > 0$ such that either $x \sqsubseteq f^n.x$ or $f^n.x \sqsubseteq x$.*

Proof We will prove only the first item, the other two are left to the reader.

1. Let A be the upper fringe of P. Assume it is finite, say with cardinality equal to k for $k > 0$ as A is nonempty. Take $a \in A$ and consider the set $S = \{f^n.a | 0 < n \leq k + 1\}$ (clearly nonempty). If $S \cap (P \backslash A) \neq \emptyset$ then there exists an $f^n.a \in S$ such that $f^n.a \in P \backslash A$. But A is the upper fringe of P and hence $f^n.a \sqsubseteq a$. Otherwise $S \cap (P \backslash A) = \emptyset$, that is, $S \subseteq A$, and hence the cardinality of S is less than k. But this means that there exist $m < n \leq k + 1$ such that $f^m.a = f^n.a \in A$, and hence $f^{n-m}.f^m.a \sqsubseteq f^m.a$.

□

Let P, Q be two partial orders. Then P^{op} is a partial order with the reverse order (with respect to P), $P \times Q$ 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 $x_1, x_2 \in f^{-1}.y$, $x_1 \sqsubseteq x_2 \Leftrightarrow x_1 \sqsubseteq_P x_2$. If A is the upper (lower) fringe of P then A is the lower (upper) fringe of P^{op}, while if A is an antichain of P then it is also an antichain of P^{op}.

4.1 Fixed Points

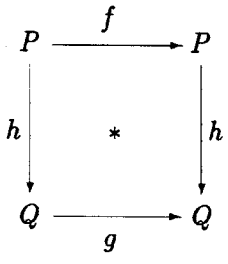
For any partial order P, function $f : P \to P$ and ordinal λ, define $f^{<\lambda>} \in P$ by

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

Of course $f^{<\lambda>}$ need not to exist, since $\bigsqcup_{k<\lambda} f^{<k>}$ need not to exist. Note that $f^{<0>} = f.\bot$ when the least element $\bot$ 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 $f^{<\lambda>}$ always exists and moreover, $(f^{<\lambda>})_\lambda$ stabilizes [HP72]. If additionally f is continuous then it has closure ordinal $\leq \omega$.

The following theorem, taken from [AP81, 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 4.4 *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:*

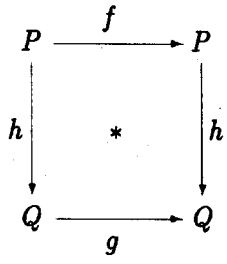


Then if $f^{<\lambda>}$ exists so does $g^{<\lambda>}$, and indeed $g^{<\lambda>} = h.f^{<\lambda>}$. In particular if $\mu.f$ exists (being an $f^{<\lambda>}$) then so does $\mu.g$ and $\mu.g = h.\mu.f$.

Several generalizations (always with monotone functions making the diagram above commutes) and applications of the theorem above, often called transfer lemma, can be found in [Mey85].

The next theorem shows that we can drop the condition of g to be monotone provided that h satisfies some extra conditions and P is a complete partial order:

Theorem 4.5 *Let $(P, \sqsubseteq_P)$ be a complete partial order and $(Q, \sqsubseteq_Q)$ partial order, and $f : P \to P$ be a monotone function, $g : Q \to Q$ be a function and $h : P \to Q$ be an onto and continuous function such that the following diagram commutes:*



Then $g^{<\lambda>}$ exists, and indeed $g^{<\lambda>} = h.f^{<\lambda>}$. Moreover, if for each $y \in Q$ the partial order determined by $h^{-1}.y$ has either

- *the finite upper fringe or*
- *the finite lower fringe or*
- *only finite antichains,*

then $\mu.g$ exists and $\mu.g = h.\mu.f$.

Proof The proof contains part of the proof of the Theorem 4.4 [AP86]: suppose $f^{<\lambda>}$ exists (as P is a complete partial order $f^{<\lambda>}$ is always defined for each ordinal λ). First we prove $h.f^{<\lambda>} = g^{<\lambda>}$ for each ordinal λ:

$$h.f^{<\lambda>}$$

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

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

= { commutativity of the diagram }

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

= { since h is continuous and strict (by lemma 4.2 h is monotone and as it is also onto then h is strict) }

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

= { induction hypothesis }

$$g.\bigsqcup_{k<\lambda} g^{<k>}$$

$= \{$ definition of $g^{<\lambda>}$ $\}$

$$g^{<\lambda>}.$$

Note that the continuity and strictness of h are essential for the existence of $\bigsqcup_{k<\lambda} g^{<k>}$.

Let now $\mu.f = f^{<\alpha>}$ for some ordinal α. We have

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

So $g^{<\alpha>}$ is a fixed point of g. In [AP86] this is enough to prove that $g^{<\alpha>} = \mu.g$ because g is monotone. In our case, we have to prove it.

Let $y \in Q$ such that $g.y = y$ and consider the partial order generated by $h^{-1}.y$. There are three cases:

- $h^{-1}.y$ has the finite upper fringe A.

  Then by lemma 4.3 there exist $a \in A$ and a natural number $n > 0$ such that $f^n.a \sqsubseteq a$.

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

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

  $\lambda > 0)$ $\{$ 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 is monotone and hence $f^{<\lambda>}$ is monotone in λ $\}$

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

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

  $$\bigsqcup_{k<\lambda+1} f^{<k>} \sqsubseteq f.a$$

  $\Rightarrow \{$ f is monotone $\}$

  $$f.\bigsqcup_{k<\lambda+1} f^{<k>} \sqsubseteq f^2.a$$

  $\Rightarrow \{$ definition of $f^{<\lambda+1>}$ $\}$

  $$f^{<\lambda+1>} \sqsubseteq f^2.a$$

$\Rightarrow$ { iterating this procedure }

$f^{<\lambda+n-1>} \sqsubseteq f^n.a$

$\Rightarrow$ { $f^{<\lambda>} \sqsubseteq f^{<\lambda+n-1>}$ }

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

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

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

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

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

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

- $h^{-1}.y$ has the finite lower fringe A

Then by lemma 4.3 there exists an $a \in A$ and a natural number $n > 0$ such that $a \sqsubseteq f^n.a$. Define for each ordinal λ, $\tilde{f}^{<\lambda>} \in P$ by

$$\tilde{f}^{<\lambda>} = \begin{cases} a & \lambda = 0 \\ f^n.\bigsqcup_{k<\lambda} \tilde{f}^{<k>} & \text{otherwise.} \end{cases}$$

Note that $\tilde{f}^{<\lambda>}$ is always defined since P is a complete partial order and $\{\tilde{f}^{<k>} | k < \lambda\}$ is a directed set, indeed by transfinite induction we have that $\tilde{f}^{<\lambda>} \sqsubseteq \tilde{f}^{<\lambda+1>}$:

$\lambda = 0$) $\tilde{f}^{<0>} = a \sqsubseteq f^n.a = \tilde{f}^{<1>}$

$\lambda > 0$) { induction hypothesis }

$(\forall k < \lambda : \tilde{f}^{<k>} \sqsubseteq \tilde{f}^{<k+1>})$

$\Rightarrow$ { $\bigsqcup$ is monotone }

$$\bigsqcup_{k<\lambda} \tilde{f}^{<k>} \sqsubseteq \bigsqcup_{k<\lambda} \tilde{f}^{<k+1>}$$

$\Rightarrow$ { $\bigsqcup_{k<\lambda} \tilde{f}^{<k+1>} = \bigsqcup_{k<\lambda+1} \tilde{f}^{<k>}$ }

$$\bigsqcup_{k<\lambda} \tilde{f}^{<k>} \sqsubseteq \bigsqcup_{k<\lambda+1} \tilde{f}^{<k>}$$

$\Rightarrow$ { f monotone implies f^n monotone }

$$f^n.\bigsqcup_{k<\lambda} \tilde{f}^{<k>} \sqsubseteq f^n.\bigsqcup_{k<\lambda+1} \tilde{f}^{<k>}$$

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

$\tilde{f}^{<\lambda>} \sqsubseteq \tilde{f}^{<\lambda+1>}$.

Using transfinite induction again, we prove now $h.\tilde{f}^{<\lambda>} = y$ for each ordinal λ:

$\lambda = 0$) $h.\tilde{f}^{<0>} = h.a = y$

$\lambda > 0$) $h.\tilde{f}^{<\lambda>}$

= { definition of $\tilde{f}^{<\lambda>}$ }

$$h.f^n.\bigsqcup_{k<\lambda}\tilde{f}^{<k>}$$

=

$$h.f.f^{n-1}.\bigsqcup_{k<\lambda}\tilde{f}^{<k>}$$

= { commutativity of the diagram }

$$g.h.f^{n-1}.\bigsqcup_{k<\lambda}\tilde{f}^{<k>}$$

= { iterating n times the last step }

$$g^n.h\bigsqcup_{k<\lambda}\tilde{f}^{<k>}$$

= { h is continuous }

$$g^n.\bigsqcup_{k<\lambda}h.\tilde{f}^{<k>}$$

= { induction hypothesis }

$$g^n.\bigsqcup_{k<\lambda}y$$

=

$$g^n.y$$

= { y is a fixed point of g }

$$y.$$

Moreover, for each ordinal λ we have $f^{<\lambda>} \sqsubseteq \tilde{f}^{<\lambda>}$; by transfinite induction:

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

$\lambda > 0$) { induction hypothesis }

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

$\Rightarrow$ {$\bigsqcup$ is monotone }

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

$\Rightarrow$ { f is monotone }

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

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

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

$\Rightarrow$ { f monotone implies $f^{<\lambda>}$ monotone in λ }

$$(\forall k \leq \lambda \ : \ f^{<k>} \sqsubseteq f.\bigsqcup_{k<\lambda}\tilde{f}^{<k>})$$

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

$$\bigsqcup_{k<\lambda+1} f^{<k>} \sqsubseteq f.\bigsqcup_{k<\lambda}\tilde{f}^{<k>}$$

$\Rightarrow$ { f is monotone }

$$f.\bigsqcup_{k<\lambda+1} f^{<k>} \sqsubseteq f^2.\bigsqcup_{k<\lambda} \tilde{f}^{<k>}$$

$\Rightarrow$ { definition of $f^{<\lambda+1>}$ }

$$f^{<\lambda+1>} \sqsubseteq f^2.\bigsqcup_{k<\lambda} \tilde{f}^{<k>}$$

$\Rightarrow$ { iterating the last 5 steps }

$$f^{<\lambda+n-1>} \sqsubseteq f^n.\bigsqcup_{k<\lambda} \tilde{f}^{<k>}$$

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

$$f^{<\lambda+n-1>} \sqsubseteq \tilde{f}^{<\lambda>}$$

$\Rightarrow$ { f monotone implies $f^{<\lambda>}$ monotone in λ }

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

Finally we have, applying this last result to the ordinal α:

$$f^{<\alpha>} \sqsubseteq \tilde{f}^{<\alpha>}$$

$\Rightarrow$ { h is monotone }

$$h.f^{<\alpha>} \sqsubseteq h.\tilde{f}^{<\alpha>}$$

$\Rightarrow$ { definition of $g^{<\alpha>}$ }

$$g^{<\alpha>} \sqsubseteq h.\tilde{f}^{<\alpha>}$$

$\Rightarrow$ { $h.\tilde{f}^{<\alpha>} = y$ }

$$g^{<\alpha>} \sqsubseteq y.$$

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

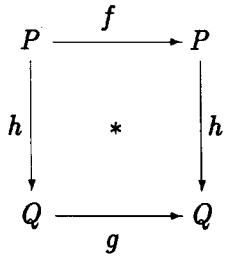
- Every antichain in $h^{-1}.y$ is finite.

  Then by lemma 4.3 there exists $a \in h^{-1}.y$ and a natural number $n > 0$ such that $f^n.a \sqsubseteq a$ or $a \sqsubseteq f^n.a$. In the first case the proof is as in the case when $h^{-1}.y$ has the finite upper fringe, otherwise is as in the case when $h^{-1}.y$ has the finite lower fringe.

□

Note that even if g is not monotone, the theorem above ensures the existence of a least fixed point for g and moreover, it can be obtained by iteration, since for all ordinals λ, g^λ exists. With a similar proof we have that P need not to be a complete partial order and h to be continuous for the existence of $\mu.g$ if for all $y \in Q$ the upper fringe of $h^{-1}.y$ exists and is finite. This does not holds in case the $h^{-1}.y$ has the finite lower fringe or only finite antichain because we need both the completeness of P and the continuity of h to prove $h.\mu.f$ be the least fixed point of g: without these two conditions we can only prove $h.\mu.f$ be a fixed point of g.

Theorem 4.6 *Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be two partial orders, and $f : P \to P$ be a monotone function, $g : Q \to Q$ be a function and $h : P \to Q$ be an onto and monotone function such that for all $y \in Q$ the upper fringe of $h^{-1}.y$ exists and is finite and the following diagram commutes:*



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 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 $g.y = y$, as $h^{-1}.y$ as the upper fringe A, then there exists an $a \in A \subseteq h^{-1}.y \subseteq P$ and a natural number $n > 0$ such that $f^n.a \sqsubseteq a$. Let us prove $f^n.a \in h^{-1}.y$:

$$h.f^n.a = h.f.f^{n-1}.a = g.h.f^{n-1}.a = \ldots = g^n.y = y.$$

by commutativity of the diagram and as y is a fixed point for g.

Now we can prove by transfinite induction $f^{<\lambda>} \sqsubseteq a$ for each ordinal λ in the same way as we did in Theorem 4.5 in the case h^{-1} has the finite upper fringe, obtaining that also $f^{<\alpha>} \sqsubseteq a$ and hence 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 h is also continuous, then $h.f^{<\lambda>} = g^{<\lambda>}$ as we have already proved in the first part of the Theorem 4.5. □

In the following we present a number of examples which show that the condition in the Theorem 4.5 saying that the function $h : P \to Q$ is onto, continuous and for each $y \in Q$ the partial order determined by $h^{-1}.y$ has either the finite upper fringe or the finite lower fringe or only finite antichains cannot be weakened in any obvious way. Of course when we consider a non-onto function h we have to consider also a non-monotone function $g : Q \to Q$ according to the theorem 4.4, while when we consider non-continuous but monotone functions h we have to take only those such that for each $y \in Q$ the partial order determined by $h^{-1}.y$ has not the upper fringe according to the Theorem 4.6

- Let P be the flat domain $\{x\}_\perp$ and Q be the flat domain $\{a, b\}_\perp$. Consider the following three functions $f : P \to P, g : Q \to Q$ and $h : P \to Q$:

$$\begin{array}{lll} f.\perp = x & g.\perp = a & h.\perp = \perp \\ f.x = x & g.a = a & h.x = a. \\ & g.b = b & \end{array}$$

Then f is monotone and has least fixed point x, h is monotone, strict, *non-onto* and for each $y \in Q$, $h^{-1}.y$ has the upper fringe, the lower fringe and every antichain finite, even if g (non-monotone) makes the diagram of the theorem commutes, then g still has two incomparable fixed points, a and b.

- Let $P = \{x, y, \bot\}$ be the complete partial order with $\bot \sqsubseteq x \sqsubseteq y$ and let Q be the flat domain $\{a, b\}_\bot$.

  Consider the following functions $f : P \to P, g : Q \to Q$ and $h : P \to Q$:

  $$\begin{array}{lll} f.\bot = x & g.\bot = a & h.\bot = \bot \\ f.x = x & g.a = a & h.x = a \\ f.y = y & g.b = b & h.y = b. \end{array}$$

  Then f is monotone and has least fixed point x, h is strict, onto, *non-monotone* ($x \sqsubseteq y$ but $h.x = a \not\sqsubseteq b = h.y$), *non-continuous* ($y = \bigsqcup\{x, y\}$ but $h.y = a \neq \bigsqcup\{h.x, h.y\}$) and for each $z \in Q$, $h^{-1}.z$ has the finite upper fringe, the finite lower fringe and every antichain is finite. Even if g (non-monotone) makes the diagram commutes then g still has two different and incomparable fixed points, a and b.

- Let $P = \{x_i | i \geq 0\} \cup \{x_\omega\}$ be the complete partial order ordered by:

  $$(\forall i \leq j \ : \ x_i \sqsubseteq x_j) \ \wedge \ (\forall i \geq 0 \ : \ x_i \sqsubseteq x_\omega) \ \wedge \ x_\omega \sqsubseteq x_\omega.$$

  and let Q be the flat domain $\{a\}_\bot$. Consider the following functions $f : P \to P, g : Q \to Q$ and $h : P \to Q$:

  $$\begin{array}{lll} f.x_i = x_{i+1} & g.\bot = \bot & h.x_i = \bot \\ f.x_\omega = x_\omega & g.a = a & h.x_\omega = a. \end{array}$$

  Then f is monotone and has least fixed point x_ω, h is onto, monotone, *non-continuous* ($x_\omega = \bigsqcup x_i$ but $h.x_\omega = a \neq \bot = \bigsqcup \bot = \bigsqcup h.x_i$) and for each $z \in Q$, $h^{-1}.z$ has the finite lower fringe and every antichain is finite. Even if g makes the diagram commutes then g still has least fixed point $\bot$ but $\mu.g = \bot \neq a = h.x_\omega = h.\mu.f$. Note that there is no upper fringe according to the Theorem 4.6.

- Let $P = \{x_i | i \geq 0\} \cup \{x_\omega, x_{\omega+1}\} \cup \{y_i | i \geq 0\}$ be the complete partial order ordered by:

  $$\begin{array}{ll} (\forall i \leq j \ : & x_i \sqsubseteq y_j \ \wedge \ x_i \sqsubseteq x_j), \\ (\forall i \geq 0 \ : & x_i \sqsubseteq x_\omega \ \wedge \ x_i \sqsubseteq x_{\omega+1} \ \wedge \ y_i \sqsubseteq y_i), \\ & x_\omega \sqsubseteq x_\omega \ \wedge \ x_{\omega+1} \sqsubseteq x_{\omega+1} \ \wedge \ x_\omega \sqsubseteq x_{\omega+1}, \end{array}$$

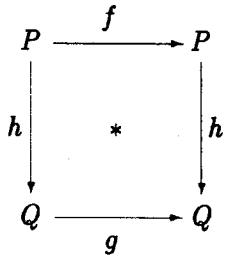
  and let Q be the flat domain $\{a, b\}_\bot$. Consider the following three functions $f : P \to P, g : Q \to Q$ and $h : P \to Q$:

  $$\begin{array}{lll} f.x_i = x_{i+1} & g.\bot = a & h.x_i = \bot \\ f.y_i = y_{i+1} & g.a = a & h.y_i = b \\ f.x_\omega = x_{\omega+1} & g.b = b & h.x_\omega = \bot \\ f.x_{\omega+1} = x_{\omega+1} & & h.x_{\omega+1} = a. \end{array}$$

  Then f is monotone and has least fixed point $x_{\omega+1}$, h is onto, monotone and continuous but $h^{-1}.\ \bot$ *has not finite upper fringe, not finite lower fringe and not all the antichain are finite.* Even if g (non monotone) makes the diagram of the theorem commutes, then g still has two incomparable fixed points, a and b.

We have seen that in Theorem 4.5 that the property of a monotone function $f : P \to P$ of having a least fixed point is transferred to a a function $g : Q \to Q$, that in general need not to be monotone (and hence continuous), via a function $h : P \to Q$. This is important, since the the existence of a "computable" function that is not monotone, violates the Scott's most reasonable thesis that all computable function are continuous, and hence monotones. However, the Theorem 4.5 shows that we can construct the least fixed point of the function g by iteration from the least element, thus we have a good reason to say that every computable function g is continuos in a subset of its co-domain which includes the set $\{g^n | n \in N\}$ of all its iterations.The next lemma determines on which subset of Q, depending from P and h, the function g is monotone:

Lemma 4.7 *Let $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ be two partial orders, and $f : P \to P$ be a monotone function, $g : Q \to Q$ be a functions and $h : P \to Q$ be an onto and monotone function such that the following diagram commutes:*



Then for all $y_1, y_2 \in Q$ we have:

$$\begin{array}{l} y_1 \sqsubseteq_Q y_2 \wedge \\ \wedge\ (\exists x_1,\ x_2 \in P\ :\ x_1 \in h^{-1}.y_1,\ x_2 \in h^{-1}.y_2\ \wedge\ x_1 \sqsubseteq_P x_2) \\ \Rightarrow g.y_1 \sqsubseteq g.y_2. \end{array}$$

Proof Let $y_1, y_2 \in Q$ and assume there exist $x_1 \in h^{-1}.y_1$ and $x_2 \in h^{-1}.y_2$ such that $x_1 \sqsubseteq_P x_2$. Then we have:

$$x_1 \sqsubseteq_P x_2$$

$\Rightarrow$ { f is monotone }

$$f.x_1 \sqsubseteq_P f.x_2$$

$\Rightarrow$ { h is monotone }

$$h.f.x_1 \sqsubseteq_Q h.f.x_2$$

$\Rightarrow$ { commutativity of the diagram }

$$g.h.x_1 \sqsubseteq_Q g.h.x_2$$

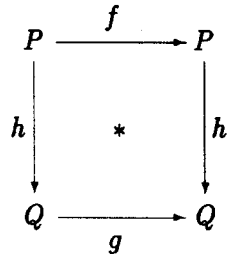
$\Rightarrow$ { $h.x_1 = y_1$ and $h.x_2 = y_2$ }

$$g.y_1 \sqsubseteq_Q g.y_2.$$

□

A similar result holds also for the transfer of the continuity property from f to g:

Lemma 4.8 *Let $(P, \sqsubseteq_P)$ be a complete partial order, $(Q, \sqsubseteq_Q)$ be partial order, and $f : P \to P$ be a continuous function, $g : Q \to Q$ be a functions and $h : P \to Q$ be an onto and continuous function such that the following diagram commutes:*



Then for all $\{y_i | i \in I\} \subseteq Q$ such that

$$\{y_i | i \in I\} \text{ is directed} \wedge$$
$$\wedge\ (\forall i \in I : \exists x_i \in P\ :\ x_i \in h^{-1}.y_i\ \wedge\ \{x_i | i \in I\} \subseteq P \text{ is directed}),$$

we have $g.\bigsqcup\{y_i | i \in I\} = \bigsqcup\{g.y_i | i \in I\}$.

Proof Let $\{y_i | i \in I\} \subseteq Q$ be directed and assume that also $\{x_i | i \in I\} \subseteq P$ is directed, where $x_i \in h^{-1}.y_i$ for each $i \in I$. As f is continuous and P is a complete partial order we have:

$$f.\bigsqcup\{x_i | i \in I\} = \bigsqcup\{f.x_i | i \in I\}$$

$\Rightarrow$

$$h.f.\bigsqcup\{x_i | i \in I\} = h.\bigsqcup\{f.x_i | i \in I\}$$

$\Rightarrow$ { h is continuous }

$$h.f.\bigsqcup\{x_i | i \in I\} = \bigsqcup\{h.f.x_i | i \in I\}$$

$\Rightarrow$ { commutativity of the diagram }

$$g.h.\bigsqcup\{x_i | i \in I\} = \bigsqcup\{g.h.x_i | i \in I\}$$

$\Rightarrow$ { h is continuous }

$$g.\bigsqcup\{h.x_i | i \in I\} = \bigsqcup\{g.h.x_i | i \in I\}$$

$\Rightarrow$ { $h.x_i = y_i$ for all $i \in I$ }

$$g.\bigsqcup\{y_i | i \in I\} = \bigsqcup\{g.y_i | i \in I\}.$$

□

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 $\{\sigma |\ P.\sigma = tt\} \subseteq \Sigma$ 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 $A = \{\sigma |\ P(A).\sigma = tt\}$ and conversely, if P is a predicate then $P = P(\{\sigma |\ P.\sigma = tt\})$.

A predicate transformer is a function in $Pred \rightarrow Pred$ which satisfies some properties. There are different definitions of predicate transformers in the literature that differs 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 numbers greater than 0 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 $P_i \in Pred$ 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 $P_i \in Pred$ for all $i \in I$ *(multiplicativity)*.

In [Dij76] a predicate transformer $\pi \in Pred \rightarrow Pred$ 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. Next we are going to relate these conditions. In order to do this we need the following lemma which is a slight variation of the stability lemma in [AP86]:

Lemma 4.9 *Let $\pi \in PTran$ satisfying the Σ-multiplicativity law, and let $\sigma \in \Sigma$ such that $\pi.\mathbf{true}.\sigma$. Then there is a set $min(\pi, \sigma) \subseteq \Sigma$ such that*

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

Proof Let $(j \in)J$ be an index set of the same cardinality as Σ and for all $j \in J$ let ρ_j be those elements of Σ such that there is a predicate Q with $Q.\rho_j = ff$ but $\pi.Q.\sigma = tt$ (If there are none we can take $min(\pi, \sigma) = \Sigma$). Let now for all $j \in J$, Q_j be a family of predicate such that $Q_j.\rho_j = ff$ but $\pi.Q_j.\sigma = tt$. Define $min(\pi, \sigma) = \bigcap_{j \in J}\{\sigma' |\ Q_j.\sigma' = tt\}$. Since π is a predicate transformer and for all $j \in J$ is $\pi.Q_j.\sigma = tt$, it follows from the Σ-multiplicativity law that $\pi.\bigwedge_{j \in J} Q_j.\sigma = tt$. But $\bigwedge_{j \in J} Q_j = \{\sigma' | (\forall j \in J\ :\ Q_j.\sigma' = tt)\} = \bigcap_{j \in J}\{\sigma' |\ Q_j.\sigma' = tt\} = min(\pi, \sigma)$, thus $\pi.min(\pi, \sigma).\sigma = tt$ (considering $min(\pi, \sigma)$ as a predicate). So if $min(\pi, \sigma) \subseteq \{\sigma' | Q.\sigma' = tt\}$

then $min(\pi,\sigma) \Rightarrow Q$, and hence $\pi.min(\pi,\sigma) \Rightarrow \pi.Q$ as π is monotone. But we have seen that $\pi.min(\pi,\sigma).\sigma = tt$, hence also $\pi.Q.\sigma = tt$.

Conversely, suppose that $\pi.Q.\sigma = tt$ but that, for sake of contradiction, $min(\pi,\sigma) \not\subseteq \{\sigma' | Q.\sigma' = tt\}$. Then there is some ρ in $min(\pi,\sigma)$ but $Q.\rho = ff$, and so ρ must be a ρ_j for some $j \in J$, and then $\rho \notin \{\sigma' |\ Q_j.\sigma' = tt\} \supseteq \bigcap_{j\in J}\{\sigma' |\ Q_j.\sigma' = tt\} = min(\pi,\sigma)$, contradicting $\rho \in min(\pi,\sigma)$. □

Note that if a predicate transformer π satisfies the law of excluded miracles, then for all the $\sigma \in \Sigma$ the set $min(\pi,\sigma)$ is non-empty.

Theorem 4.10 *Let Σ be a countable set of states with cardinality greater than 1 and let 1. - 8. be the list of properties defined above. Then we have*

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

Proof

4. $\wedge$ 5. $\Rightarrow$ 8.) A proof is given in [Bes83].

8. $\Rightarrow$ 6. $\Rightarrow$ 7.) Clearly multiplicativity implies countable multiplicativity, and this implies the Σ-multiplicativity since Σ is countable.

7. $\Rightarrow$ 8.) Let $\{P_i \in Pred | i \in I\}$ be a set of predicates on Σ where $I \neq \emptyset$ (but possibly, I is uncountable) and let π be a predicate transformer satisfying the Σ-multiplicativity law. It suffices to prove $\bigwedge_{i\in I} \pi.P_i \Rightarrow \pi.\bigwedge_{i\in I} P_i$ since the other direction is trivial (if the cardinality of Σ is greater than 1). Let $\sigma \in \Sigma$ such that $\bigwedge_{i\in I} \pi.P_i.\sigma$, then $\pi.P_i.\sigma$ for each $i \in I$ and hence by lemma 4.9 it is $min(\pi,\sigma) \subseteq \{\sigma' | P_i.\sigma'\}$ for each $i \in I$. But then $min(\pi,\sigma) \subseteq \bigcap_{i\in I}\{\sigma' | P_i.\sigma'\}$ and hence applying again the lemma 4.9 but in the other direction we obtain $\pi.\bigwedge_{i\in I} P_i.\sigma = tt$. Therefore $\bigwedge_{i\in I} \pi.P_i \Rightarrow \pi.\bigwedge_{i\in I} P_i$.

8. $\Leftrightarrow$ 6. $\Leftrightarrow$ 7.) By the two item above we have 8. $\Rightarrow$ 6. $\Rightarrow$ 7. $\Rightarrow$ 8..

8. $\Rightarrow$ 3.) Let $P, Q \in Pred$ such that $P \Rightarrow Q$. Then $P \wedge Q = P$ and as π is a predicate transformer satisfying the multiplicativity law we have:

$$\pi.P = \pi.(P \wedge Q) = \pi.P \wedge \pi.Q$$

that is, $\pi.P \Rightarrow \pi.Q$.

□

In a similar way we can prove that if Σ is uncountable then 8. $\Leftrightarrow$ 7. $\Rightarrow$ 6. $\Rightarrow$ 3.

From now on, we will consider the following definition of predicate transformers:

Definition 4.11 *A predicate transformer is any function $\pi \in PTran = Pred \rightarrow Pred$ which satisfies the multiplicativity law.*

The previous lemma together with the lemma 4.2 show that predicate transformers as defined in [Dij76] are 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 the predicate transformers in this

last one are also predicate transformers in the sense of our definition 4.11. Finally predicate transformers in the sense of our definition 4.11 are also predicate transformers in the sense of [BvW90].

Thus our definition 4.11 generalize 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] and this forces us to not consider angelic non-determinism that violates properties 5., 6., 7., and 8..

Next we are going to relate predicate transformers with powerdomains. We generalize the relationship between the Smyth powerdomain and the predicate transformers [Wan77, Plo79, Bes83, AP86, Smy83] to 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]. For future reference, we give the following commuting diagram that summarizes all the relationships between predicate transformers and discrete powerdomains that are dealt with the next three subsections:

$$\begin{array}{ccc}
PTran_N & \xrightarrow[\eta]{\cong} & ETran^{\emptyset} = (\Sigma \longrightarrow \mathcal{E}^{\emptyset}.\Sigma_{\perp}) \\
\downarrow_1 & * & e_{\Sigma} \cdot - \\
PTran_D & \xrightarrow[\gamma]{\cong} & STran^{\delta} = (\Sigma \longrightarrow \mathcal{S}^{\delta}.\Sigma_{\perp}) \\
id_{PTran} \downarrow & * & d_{\Sigma} \cdot - \\
PTran_B & \xrightarrow[\omega]{\cong} & STran^{\emptyset} = (\Sigma \longrightarrow \mathcal{S}^{\emptyset}.\Sigma_{\perp})
\end{array}$$

All the arrows of the diagram above are monotone functions (we are working in the category **PO** of partial orders with monotone functions as morphisms).

4.2.1 Smyth powerdomain with empty set

Definition 4.12 *Let $X_\perp$ be a flat domain. Then the Smyth powerdomain of $X_\perp$ (with empty set), is defined as the partial order*

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

ordered by the superset order, that is

$$A \sqsubseteq B \ \Leftrightarrow\ A \supseteq B.$$

This definition differs from the original definition of Smyth's powerdomain [Smy78] because we add the empty set as a top element. Moreover we have no restriction on the cardinality of X.

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.

A meaning of a statement will be a function in the *Smyth State-Transformers domain*, denoted by $STran^{\emptyset}$, that 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.

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

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

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

To relate $STran^{\emptyset}$ and $PTran_B$, we define for $m \in STran^{\emptyset}$ and $Q \in Pred$ the function $\omega : STran^{\emptyset} \to PTran_B$ by

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

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

Lemma 4.13 *Let $m \in STran^{\emptyset}$. Then the function $\omega.m \in PTran_B$.*

Proof Let $P \Rightarrow Q$ and $\omega.m.P.\sigma = tt$. Then $P.m.\sigma = tt$ and as $P \Rightarrow Q$ also $Q.m.\sigma = tt$. Thus $\omega.m.Q.\sigma = tt$, that is, $\omega.m.P \Rightarrow \omega.m.Q$. Multiplicativity is clear. □

Lemma 4.14 *The function ω is monotone.*

Proof Let $m \sqsubseteq m'$ and $\omega.m.P.\sigma = tt$. Then $m'.\sigma \subseteq m.\sigma$ and as $P.m.\sigma = tt$ then $P.m'.\sigma = tt$. □

The function ω has an inverse. Define for a predicate transformer $\pi \in PTran_B$ and $\sigma \in \Sigma$ the function $\omega^{-1} : PTran_B \to STran^{\emptyset}$ by:

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

Lemma 4.15 *The function ω^{-1} is monotone.*

Proof Let $\pi \sqsubseteq_B \pi'$ and $\sigma \in \Sigma$. If $\omega^{-1}.\pi.\sigma = \Sigma_{\perp}$ then clearly $\omega^{-1}.\pi.\sigma \sqsubseteq \omega^{-1}.\pi'.\sigma$. Otherwise $\pi'.\mathbf{true}.\sigma$ because $\pi.\mathbf{true} \Rightarrow \pi'.\mathbf{true}$. So $\omega^{-1}.\pi.\sigma = min(\pi, \sigma)$ and also $\omega^{-1}.\pi'.\sigma = min(\pi', \sigma)$. Since $\pi.min(\pi, \sigma) \Rightarrow \pi'.min(\pi, \sigma)$ and $\pi.min(\pi, \sigma).\sigma = tt$ we have also $\pi'.min(\pi, \sigma).\sigma = tt$. Hence applying the stability lemma 4.9 to π' we obtain $min(\pi', \sigma) \subseteq min(\pi, \sigma)$,i.e. $\omega^{-1}.\pi.\sigma \sqsubseteq \omega^{-1}.\pi'.\sigma$. □

Finally we have:

Theorem 4.16 *The function $\omega : STran^{\emptyset} \to PTran_B$ is an isomorphism of partial orders with inverse ω^{-1}.*

Proof The previous two lemmas showed that ω and ω^{-1} are monotone, so it remains to prove that they form an isomorphism:

$\omega \circ \omega^{-1} = id_{PTran_B}$) Let $\pi \in PTran_B$ and P be a predicate; we have

$$\omega.(\omega^{-1}.\pi).P$$

$= \{$ definition of ω $\}$

$$\{\sigma | P.(\omega^{-1}.\pi).\sigma\}$$

$=\{$ definition of ω^{-1} $\}$

$$\{\sigma | \pi.\mathbf{true}.\sigma \ \wedge \ P.min(\pi, \sigma)\}$$

$= \{$ stability lemma 4.9 $\}$

$$\{\sigma | \pi.\mathbf{true}.\sigma \ \wedge \ \pi.P.\sigma = tt\}$$

$= \{ P \Rightarrow \mathbf{true} \}$

$$\{\sigma | \pi.P.\sigma = tt\}$$

$=$

$$\pi.P$$

$\omega^{-1} \circ \omega = id_{STran^{\emptyset}}$) Let $m \in STran^{\emptyset}$ and $\sigma \in \Sigma$. There are two cases:
If $m.\sigma = \Sigma_\perp$ then $\omega.m.\mathbf{true}.\sigma = ff$. So if $m.\sigma = \Sigma_\perp$ we have $\omega^{-1}.(\omega.m).\sigma = \Sigma_\perp = m.\sigma$. Otherwise $\omega^{-1}.(\omega.m).\sigma = min(\omega.m, \sigma)$. Now for a predicate P we have by definition of ω that $\omega.m.P.\sigma = tt$ if and only if $P.m.\sigma = tt$; hence when $m.\sigma \neq \Sigma_\perp$ we can consider $m.\sigma$ as a predicate and hence $\omega.m.(m.\sigma).\sigma = tt$. Since $\omega.m.(m.\sigma) = \{\sigma | (m.\sigma).m.\sigma\} = m.\sigma$ we have by stability lemma 4.9 $min(\omega.m, \sigma) = m.\sigma$. Therefore $\omega^{-1}.(\omega.m).\sigma = m.\sigma$

□

4.2.2 Smyth powerdomain with deadlock

Definition 4.17 *Let $X_\perp$ be a flat domain. Then the Smyth's powerdomain with deadlock of $X_\perp$, is defined as the partial order*

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

ordered by

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

This definition differs from both the original definition of the Smyth powerdomain [Smy78] and the definition above of Smyth's powerdomain with empty set because we have no empty set and moreover 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, for example consider in $\mathcal{S}^{\delta}.N_\perp$ the following directed set which has no upper bound:

$$N \sqsubseteq N\backslash\{0\} \sqsubseteq N\backslash\{0,1\} \sqsubseteq ...,$$

(this example is from [AP86]).

The Smyth powerdomain with deadlock and Smyth powerdomain with empty set are related by a function $d_X : \mathcal{S}^\delta.X_\perp \to \mathcal{S}^\emptyset.X_\perp$ defined by

$$d_X.A = \begin{cases} A & \text{if } A \neq \delta \\ \emptyset & \text{otherwise} \end{cases}$$

as is shown in the following lemma:

Lemma 4.18 *The function $d_X : \mathcal{S}^\delta.X_\perp \to \mathcal{S}^\emptyset.X_\perp$ is onto, continuous, and for each $A \in \mathcal{S}^\emptyset.X_\perp$ the upper fringe and the lower fringe of $d_X^{-1}.A$ exist and they are finite. Moreover, each antichain of $d_X^{-1}.A$ is finite.*

Proof It follows directly from the observation that $d_X^{-1}.A$ consists of only one element (that is hence at the same time the top, the bottom and the unique nonempty antichain). □

We will also use this lemma later in order to apply the Theorem 4.5. The next lemma is useful to apply the lemma 4.7:

Lemma 4.19 *Let $y_1, y_2 \in \mathcal{S}^\emptyset.X_\perp$. Then:*

$$y_1 \sqsubseteq y_2 \ \wedge\ (y_2 = \emptyset \Rightarrow y_1 = \Sigma_\perp \ \vee\ y_2 = \emptyset) \ \Rightarrow\ (\exists x_1 \in d_X^{-1}.y_1,\ x_2 \in d_X^{-1}.y_2 \ :\ x_1 \sqsubseteq x_2).$$

Proof Let $y_1 \sqsubseteq y_2$. If $y_2 \neq \emptyset$ then also $y_1 \neq \emptyset$ because $y_1 \sqsubseteq y_2$, thus $y_1 \in d_X^{-1}.y_1$ and $y_2 \in d_X^{-1}.y_2$. Take $x_1 = y_1$ and $x_2 = y_2$, and as $y_1 \sqsubseteq y_2$ we have $x_1 \sqsubseteq x_2$. If $y_2 = \emptyset$ then by hypothesis $y_1 = \Sigma_\perp \ \vee\ y_2 = \emptyset$. Hence take $x_1 = \delta \in d_X^{-1}.y_2$ and take x_1 either $\Sigma_\perp$ or $\delta \in d_X^{-1}.y_1$. In both the cases $x_1 \sqsubseteq x_2$. □

The *Smyth State-Transformers respecting deadlock*, are all the functions $\Sigma \to \mathcal{S}^\delta.\Sigma_\perp$, ordered pointwise. We denote this partial order $STran^\delta$.

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

$$\pi \sqsubseteq_{PD} \hat{\pi} \ \text{ if } \ \pi.\mathbf{false} \Rightarrow \hat{\pi}.\mathbf{false} \ \wedge \\ (\forall Q \in Pred \ :\ (\pi.Q \wedge \neg\pi.\mathbf{false}) \Rightarrow\ (\hat{\pi}.Q \wedge \neg\hat{\pi}.\mathbf{false})).$$

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

In order to relate $STran^\delta$ and $PTran_D$, we define for $m \in STran^\delta$ and $Q \in Pred$ the function $\gamma : STran^\delta \to PTran_D$ by

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

Lemma 4.20 *Let $m \in STran^\delta$. Then the function $\gamma.m \in PTran_D$.*

Proof Let $P \Rightarrow Q$ and $\gamma.m.P.\sigma = tt$. Then either $m.\sigma = \delta$ and hence $\gamma.m.Q.\sigma = tt$ or $P.m.\sigma = tt$ and as $P \Rightarrow Q$ also $Q.m.\sigma = tt$. Thus $\gamma.m.Q.\sigma = tt$, that is, $\gamma.m.P \Rightarrow \gamma.m.Q$. Multiplicativity is clear. □

Remark: if we would not consider the set $\{\sigma | m.\sigma = \delta\}$ in the definition of $\gamma.m.Q$ for $m \in STran^\delta$ and $Q \in Pred$ then the function $\gamma.m$ would not be multiplicative.

Lemma 4.21 *The function γ is monotone.*

Proof Let $m \sqsubseteq m'$. If $\gamma.m.\mathbf{false}.\sigma = tt.$ then $m.\sigma = \delta$ and hence also $m'.\sigma = \delta$, thus $\gamma.m.\mathbf{false} \Rightarrow \gamma.m'.\mathbf{false}$. Let now $\gamma.m.P.\sigma = tt$ and $\gamma.m.\mathbf{false}.\sigma = ff$ Then $m.\sigma \neq \Sigma_\perp$ and $m.\sigma \neq \delta$. Hence also $m'.\sigma \neq \delta$ and moreover $m'.\sigma \subseteq m.\sigma$. Hence $P.m.\sigma = tt$ implies also $P.m'.\sigma = tt$ and $(\gamma.m.P.\sigma \wedge \neg\gamma.m.\mathbf{false}.\sigma)$ implies $(\gamma.m'.P.\sigma \wedge \neg\gamma.m'.\mathbf{false}.\sigma)$ □

The function γ has an inverse. Define for a predicate transformer $\pi \in PTran_D$ and $\sigma \in \Sigma$ the function $\gamma^{-1} : PTran_D \to STran^\delta$ by:

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

Lemma 4.22 *The function γ^{-1} is monotone.*

Proof Let $\pi \sqsubseteq_D \pi'$ and $\sigma \in \Sigma$. If $\gamma^{-1}.\pi.\sigma = \Sigma_\perp$ then clearly $\gamma^{-1}.\pi.\sigma \sqsubseteq \gamma^{-1}.\pi'.\sigma$.

If $\gamma^{-1}.\pi.\sigma = \delta$ then $\pi.\mathbf{false}.\sigma = tt$. But $\pi.\mathbf{false} \Rightarrow \pi'.\mathbf{false}$ and hence $\pi'.\mathbf{false}.\sigma = tt$. Hence $\gamma^{-1}.\pi'.\sigma = \delta$.

Otherwise $\pi.\mathbf{true}.\sigma$ and $\neg\pi.\mathbf{false}.\sigma$. As $\pi \sqsubseteq_D \pi'$ we have also $\pi'.\mathbf{true}.\sigma$ and $\neg\pi'.\mathbf{false}.\sigma$. Thus $\gamma^{-1}.\pi.\sigma = min(\pi, \sigma)$ and also $\gamma^{-1}.\pi'.\sigma = min(\pi', \sigma)$. Since $\pi.min(\pi, \sigma) \Rightarrow \pi'.min(\pi, \sigma)$ and $\pi.min(\pi, \sigma).\sigma = tt$ we have also $\pi'.min(\pi, \sigma).\sigma = tt$. Hence applying the stability lemma 4.9 to π' we obtain $min(\pi', \sigma) \subseteq min(\pi, \sigma)$.

Therefore in each case $\gamma^{-1}.\pi.\sigma \sqsubseteq \gamma^{-1}.\pi'.\sigma$. □

Also in this case we have an order-isomorphism:

Theorem 4.23 *The function $\gamma : STran^\delta \to PTran_D$ is an isomorphism of partial orders with inverse γ^{-1}.*

Proof The previous two lemmas showed that γ and γ^{-1} are monotone, so it remains to prove that they form an isomorphism:

$\gamma \circ \gamma^{-1} = id_{PTran_D}$) Let $\pi \in PTran_D$ and and P be a predicate; we have

$$\gamma.(\gamma^{-1}.\pi).P$$

$$= \{ \text{ definition of } \gamma \}$$

$$\{\sigma | P.(\gamma^{-1}.\pi).\sigma\} \cup \{\sigma | (\gamma^{-1}.\pi).\sigma = \delta\}$$

$$= \{ \text{ definition of } \gamma^{-1} \}$$

$$\{\sigma | \pi.\mathbf{true}.\sigma \wedge \neg\pi.\mathbf{false}.\sigma \wedge P.min(\pi, \sigma)\} \cup \{\sigma | \pi.\mathbf{false}.\sigma\}$$

$$= \{ \text{ stability lemma 4.9 } \}$$

$$\{\sigma | \pi.\mathbf{true}.\sigma \wedge \neg\pi.\mathbf{false}.\sigma \wedge \pi.P.\sigma = tt\} \cup \{\sigma | \pi.\mathbf{false}.\sigma\}$$

$= \{ P \Rightarrow \mathbf{true} \}$

$\{\sigma | \pi.P.\sigma = tt \ \wedge \ \neg\pi.\mathbf{false}.\sigma\} \cup \{\sigma | \pi.\mathbf{false}.\sigma\}$

$=$

$\pi.P.$

$\gamma^{-1} \circ \gamma = id_{STran^\delta}$) Let $m \in STran^\delta$ and $\sigma \in \Sigma$. We have three cases: If $m.\sigma = \Sigma_\perp$ then $\gamma.m.\mathbf{true}.\sigma = ff$. Thus $\gamma^{-1}.(\gamma.m).\sigma = \Sigma_\perp = m.\sigma$.

If $m.\sigma = \delta$ then $\gamma.m.\mathbf{false}.\sigma = tt$. Thus $\omega^{-1}.(\omega.m).\sigma = \delta = m.\sigma$.

Otherwise $\gamma^{-1}.(\gamma.m).\sigma = min(\gamma.m, \sigma)$. Now for a predicate P we have by definition of γ that $\gamma.m.P.\sigma = tt$ if and only if $P.m.\sigma = tt$ (note that $m.\sigma \neq \delta$); hence when $m.\sigma \neq \Sigma_\perp$ we can consider $m.\sigma$ as a predicate and thus by stability lemma 4.9 $min(\gamma.m, \sigma) = m.\sigma$. Therefore $\gamma^{-1}.(\gamma.m).\sigma = m.\sigma$.

□

4.2.3 Egli-Milner powerdomain with empty set

Definition 4.24 *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 subsets 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 considered as an element added by means of a smash product following the ideas of [HP79, MM79, Plo81, Abr91], that is, 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: $\{\perp\}$ is the least element and if $\mathcal{F} \subseteq \mathcal{E}^\emptyset.X_\perp$ is a directed family then $\bigsqcup \mathcal{F} = (\bigcup \mathcal{F} \backslash \{\perp\}) \cup \{\perp \ | (\forall A \in \mathcal{F} \ : \perp \in A)\}$.

The Egli-Milner powerdomain with empty set and the 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 is shown in the following lemma:

Lemma 4.25 *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 upper fringe and the lower fringe of $e_X^{-1}.B$ exist and they are finite.*

Proof It is easily proved that $d_X^{-1}.B$ has the finite upper fringe since it has the top element either B itself if $B \neq \delta$ or $\emptyset$ otherwise. Moreover, it has also the finite lower fringe since it has the bottom element, that is, $\emptyset$ if $B = \delta$, $\perp$ if $B = X_\perp$ or B itself otherwise. □

We also have the following lemma that relates the Egli-Milner powerdomain with the Smyth powerdomain with empty set:

Lemma 4.26 *The function $d_X \circ e_X : \mathcal{E}^\emptyset.X_\perp \to \mathcal{S}^\emptyset.X_\perp$ is onto, continuous. Moreover, for each $B \in \mathcal{S}^\emptyset.X_\perp$ the upper fringe and the lower fringe of $(d_X.e_X)^{-1}.B$ exist and they are finite.*

We will also use later these lemmas in order to apply the Theorem 4.5. The next two lemmas are useful to apply the lemma 4.7:

Lemma 4.27 *Let $y_1, y_2 \in \mathcal{S}^\delta.X_\perp$. Then:*

$$y_1 \sqsubseteq y_2 \ \wedge \ (y_1 = \Sigma_\perp \ \vee \ y_1 = y_2) \Rightarrow (\exists x_1 \in e_X^{-1}.y_1, \ x_2 \in e_X^{-1}.y_2 \ : \ x_1 \sqsubseteq x_2).$$

Proof Let $y_1 \sqsubseteq y_2$. If $y_1 = \Sigma_\perp$ then take $x_1 = \perp$, hence $x_1 \in e_X^{-1}.y_1$ and for all $x_2 \in d_X^{-1}.y_2$ we have $x_1 \sqsubseteq x_2$. If instead $y_1 = y_2$, then take $x_1 = x_2 \in e_X^{-1}.y_1$ so that $x_1 \sqsubseteq x_2$. □

Lemma 4.28 *Let $y_1, y_2 \in \mathcal{S}^\emptyset.X_\perp$. Then:*

$$y_1 \sqsubseteq y_2 \ \wedge \ (y_1 = \Sigma_\perp \ \vee \ y_1 = y_2)) \Rightarrow (\exists x_1 \in (d_X.e_X)^{-1}.y_1, \ x_2 \in (d_X.e_X)^{-1}.y_2 \ : \ x_1 \sqsubseteq x_2).$$

Proof Similar as the proof of the previous lemma. □

The *Egli-Milner State-Transformers* are all the functions $\Sigma \to \mathcal{E}^\emptyset.\Sigma_\perp$ ordered pointwise. We will denote them by $ETran^\emptyset$. Note that in this case the 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 so-called Nelson predicate transformers [Nel87], that are introduced in the next definition:

Definition 4.29 *Define the Nelson's predicate transformers $PTran_N$ to be the set of all the functions $\pi \in Pred \to Pred \times Pred$ such that:*

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

where $\downarrow_i$: $PTran_N \to PTran$ is a projection operator defined by

$$\downarrow_i .\pi.P = P_1 \Leftrightarrow \pi.P = (P_1, P_2)$$

for each $P \in Pred$ and $i \in \{1, 2\}$. 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

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

for all $Q \in Pred$.

Moreover, $\downarrow_1$ is also monotone and continuous if we consider $\downarrow_1$ as a function from $PTran_N$ to $PTran_D$. Notice that the identity function on $PTran$ is also a monotone and continuous function from $PTran_D$ to $PTran_B$ (while this is not true in the other direction), thus $\downarrow_1$ is also a monotone and continuous function from $PTran_N$ to $PTran_B$.

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 while the order $\sqsubseteq_{PN}$ is the lifting of $\sqsubseteq_N$ to $PTran_N$.

Lemma 4.30 *Let $\pi, \rho \in PTran_N$ and define their composition by*

$$(\pi \circ \rho).Q = (\downarrow_1.\pi.\downarrow_1.\rho.Q, \downarrow_2.\pi.\downarrow_2.\rho.Q)$$

for all $Q \in Pred$. Then $\pi \circ \rho \in PTran_N$.

Proof Clearly $\downarrow_1.\pi.\downarrow_1.\rho$ and $\downarrow_2.\pi.\downarrow_2.\rho$ are predicate transformers since they are composition of predicate transformers. Let now $Q \in Pred$, then we have:

$$\downarrow_1.(\pi \circ \rho).\mathbf{true} \wedge \downarrow_2.(\pi \circ \rho).Q$$

$= \{$ definition of $(\pi \circ \rho)$ $\}$

$$\downarrow_1.\pi.\downarrow_1.\rho.\mathbf{true} \wedge \downarrow_2.\pi.\downarrow_2.\rho.Q$$

$= \{ \downarrow_1.\pi.P = \downarrow_1.\pi.\mathbf{true} \wedge \downarrow_2.\pi.P$ since $\pi \in PTran_N \}$

$$(\downarrow_1.\pi.\mathbf{true} \wedge \downarrow_2.\pi.\downarrow_1.\rho.\mathbf{true}) \wedge \downarrow_2.\pi.\downarrow_2.\rho.Q$$

$=$

$$\downarrow_1.\pi.\mathbf{true} \wedge (\downarrow_2.\pi.\downarrow_1.\rho.\mathbf{true} \wedge \downarrow_2.\pi.\downarrow_2.\rho.Q)$$

$= \{ \downarrow_2 \pi \in PTran \}$

$$\downarrow_1.\pi.\mathbf{true} \wedge \downarrow_2.\pi.(\downarrow_1.\rho.\mathbf{true} \wedge \downarrow_2.\rho.Q)$$

$= \{ \downarrow_1.\rho.\mathbf{true} \wedge \downarrow_2.\rho.P = \downarrow_1.\rho.P$ since $\rho \in PTran_N \}$

$$\downarrow_1.\pi.\mathbf{true} \wedge \downarrow_2.\pi.(\downarrow_1.\rho.Q)$$

$= \{ \downarrow_1 .\pi.\mathbf{true} \wedge \downarrow_2 .\pi.P = \downarrow_1 .\pi.P \text{ since } \pi \in PTran_N \}$

$\downarrow_1 .\pi.(\downarrow_1 .\rho.Q)$

$=$

$\downarrow_1 .(\pi \circ \rho).Q.$

Moreover, we have:

$\downarrow_2 .(\pi \circ \rho).\mathbf{true}$

$= \{$ definition of $(\pi \circ \rho)$ $\}$

$\downarrow_2 .\pi. \downarrow_2 .\rho.\mathbf{true}$

$= \{ \rho \in PTran_N \}$

$\downarrow_2 .\pi.\mathbf{true}$

$= \{ \pi \in PTran_N \}$

$\mathbf{true}.$

□

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

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

Lemma 4.31 *Let $m \in ETran^{\emptyset}$. Then the function $\eta.m \in PTran_N$.*

Proof We have to prove the following four properties:

1. $\downarrow_1 .\eta.m \in PTran.$

   Let $P \Rightarrow Q$ and $\downarrow_1 .\eta.m.P.\sigma = tt$. Then $P.m.\sigma = tt$ and as $P \Rightarrow Q$ also $Q.m.\sigma = tt$. Thus $\downarrow_1 .\eta.m.Q.\sigma = tt$, that is, $\downarrow_1 .\eta.m.P \Rightarrow \downarrow_1 .\eta.m.Q$. Multiplicativity is clear.

2. $\downarrow_2 .\eta.m \in PTran.$

   As above.

3. $(\forall P \in Pred : \downarrow_1 .\eta.m.\mathbf{true} \wedge \downarrow_2 .\eta.m.P \Leftrightarrow \downarrow_1 .\eta.m.P).$

   Let $\downarrow_1 .\eta.m.P.\sigma = tt$. Then $P.m.\sigma = tt$ and hence also $P.(m.\sigma \backslash \{\bot\}) = tt$. Thus $\downarrow_2 .\eta.m.P.\sigma = tt$ and of course $\downarrow_1 .\eta.m.\mathbf{true}.\sigma = tt$. On the other hand let $\downarrow_2 .\eta.m.P.\sigma = tt$ and $\downarrow_1 .\eta.m.\mathbf{true}.\sigma = tt$. Then $P.(m.\sigma \backslash \{\bot\}) = tt$ and $\bot \notin m.\sigma$. Thus $P.m.\sigma = tt$ and hence $\downarrow_1 .\eta.m.P.\sigma = tt$.

4. $\downarrow_2 .\eta.m.\mathbf{true} = \mathbf{true}$.

   Clear.

□

Lemma 4.32 *The function η is monotone.*

Proof Let $m \sqsubseteq m'$, $P \in Pred$. Take $\sigma \in \Sigma$ such that $\downarrow_1 .\eta.m.P.\sigma = tt$. Then $\bot \notin m.\sigma$ and $P.m.\sigma = tt$. But $m.\sigma = m'.\sigma$ thus also $P.m'.\sigma = tt$ and by definition of η, $\downarrow_1 .\eta.m'.P.\sigma = tt$, that is $\downarrow_1 .\eta.m.P \Rightarrow \downarrow_1 .\eta.m'.P$.

Take now $\sigma \in \Sigma$ such that $\downarrow_2 .\eta.m'.P.\sigma = tt$ and hence $P.(m'.\sigma \backslash \{\bot\}) = tt$ (note that now we consider m'). If $\bot \notin m.\sigma$ then $m.\sigma = m'.\sigma$ and hence also $P.(m.\sigma \backslash \{\bot\}) = tt$ and by definition of η, $\downarrow_2 .\eta.m.P.\sigma = tt$.

Otherwise $\bot \in m.\sigma$ and hence $m.\sigma \subseteq m'.\sigma$. Thus $P.(m.\sigma \backslash \{\bot\}) = tt$ and by definition of η, $\downarrow_2 .\eta.m.P.\sigma = tt$.

Therefore $\downarrow_2 .\eta.m'.P \Rightarrow \downarrow_2 .\eta.m.P$. □

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

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

Lemma 4.33 *The function η^{-1} is monotone.*

Proof Let $\pi \sqsubseteq_N \pi'$ and $\sigma \in \Sigma$. We have two cases:

- $\bot \notin \eta^{-1}.\pi.\sigma$) Then $\downarrow_1 .\pi.\mathbf{true}.\sigma = tt$ by definition of η^{-1} and as $\pi \sqsubseteq_N \pi'$ we have also $\downarrow_1 .\pi'.\mathbf{true}.\sigma = tt$. Thus $\eta^{-1}.\pi.\sigma = min(\downarrow_2 .\pi, \sigma)$ and also $\eta^{-1}.\pi'.\sigma = min(\downarrow_2 .\pi', \sigma)$. But $\downarrow_1 .\pi.\mathbf{true}.\sigma = tt$ and $\downarrow_2 .\pi.min(\downarrow_2 .\pi, \sigma).\sigma = tt$ thus as π is a Nelson predicate transformer we have $\downarrow_1 .\pi.min(\downarrow_2 .\pi, \sigma).\sigma = tt$. Moreover, $\downarrow_1 .\pi.min(\downarrow_2 .\pi, \sigma) \Rightarrow \downarrow_1 .\pi'.min(\downarrow_2 .\pi, \sigma)$ because $\pi \sqsubseteq_N \pi'$, and since π' is a Nelson predicate transformer we have also $\downarrow_1 .\pi'.min(\downarrow_2 .\pi, \sigma) \Rightarrow \downarrow_2 .\pi'.min(\downarrow_2 .\pi, \sigma)$. Thus $\downarrow_2 .\pi'.min(\downarrow_2 .\pi, \sigma).\sigma = tt$, and hence by lemma 4.9 applied to $\downarrow_2 .\pi'$ we obtain $min(\downarrow_2 .\pi', \sigma) \subseteq min(\downarrow_2 .\pi, \sigma)$.

  But $\downarrow_2 .\pi'.min(\downarrow_2 .\pi', \sigma) \Rightarrow \downarrow_2 .\pi.min(\downarrow_2 .\pi', \sigma)$ and $\downarrow_2 .\pi'.min(\downarrow_2 .\pi', \sigma).\sigma = tt$ thus also $\downarrow_2 .\pi.min(\downarrow_2 .\pi', \sigma).\sigma = tt$, and by lemma 4.9 applied this time to $\downarrow_2 .\pi$ we obtain $min(\downarrow_2 .\pi, \sigma) \subseteq min(\downarrow_2 .\pi', \sigma)$.

  Therefore if $\bot \notin \eta^{-1}.\pi.\sigma$ we have

$$\eta^{-1}.\pi.\sigma = min(\downarrow_2 .\pi, \sigma) = min(\downarrow_2 .\pi', \sigma) = \eta^{-1}.\pi'.\sigma.$$

- $\bot \in \eta^{-1}.\pi.\sigma$) As $\downarrow_2 .\pi'$ is a predicate transformer we have $\downarrow_2 .\pi'.min(\downarrow_2 .\pi', \sigma).\sigma = tt$, and as $\pi \sqsubseteq_N \pi'$ we have also $\downarrow_2 .\pi'.min(\downarrow_2 .\pi', \sigma) \Rightarrow \downarrow_2 .\pi.min(\downarrow_2 .\pi', \sigma)$. Thus $\downarrow_2 .\pi.min(\downarrow_2 .\pi', \sigma).\sigma = tt$ and hence by lemma 4.9 applied to $\downarrow_2 .\pi$ we have $min(\downarrow_2 .\pi, \sigma) \subseteq min(\downarrow_2 .\pi', \sigma)$.

  Therefore if $\bot \in \eta^{-1}.\pi.\sigma$ we have

$$\eta^{-1}.\pi.\sigma\backslash\{\bot\} = min(\downarrow_2 .\pi, \sigma) \subseteq min(\downarrow_2 .\pi', \sigma) \subseteq \eta^{-1}.\pi'.\sigma.$$

□

Finally we have:

Theorem 4.34 *The function $\eta : ETran^{\emptyset} \rightarrow PTran_N$ is an isomorphism of partial orders with inverse η^{-1}.*

Proof The previous two lemmas showed that η and η^{-1} are monotone, so it remains to prove that they form an isomorphism:

$\eta \circ \eta^{-1} = id_{PTran_N}$) Let $\pi \in PTran_N$ and and P be a predicate; we have

$$\eta.(\eta^{-1}.\pi).P$$

= { definition of η }

$$(\{\sigma | P.(\eta^{-1}.\pi).\sigma\}, \{\sigma | P.((\eta^{-1}.\pi).\sigma\backslash\{\bot\}) \})$$

= { definition of η^{-1} }

$$\begin{array}{l}(\{\sigma |\downarrow_1 .\pi.\mathbf{true}.\sigma \ \wedge \ P.min(\downarrow_2 .\pi, \sigma)\}\cup \\ \cup\{\sigma |\neg \downarrow_1 .\pi.\mathbf{true}.\sigma \ \wedge \ P.(min(\downarrow_2 .\pi, \sigma)\cup\{\bot\}) \}, \\ \{\sigma |\downarrow_1 .\pi.\mathbf{true}.\sigma \ \wedge \ P.(min(\downarrow_2 .\pi, \sigma)\backslash\{\bot\})\}\cup \\ \cup\{\sigma |\neg \downarrow_1 .\pi.\mathbf{true}.\sigma \ \wedge \ P.(min(\downarrow_2 .\pi, \sigma)\cup\{\bot\}\backslash\{\bot\}) \})\end{array}$$

= { $\bot \notin min(\downarrow_2 .\pi, \sigma)$ and $P.\bot = \mathbf{false}$ }

$$\begin{array}{l}(\{\sigma |\downarrow_1 .\pi.\mathbf{true}.\sigma \ \wedge \ P.min(\downarrow_2 .\pi, \sigma)\}, \\ \{\sigma |\downarrow_1 .\pi.\mathbf{true}.\sigma \ \wedge \ P.min(\downarrow_2 .\pi, \sigma)\}\cup \\ \cup\{\sigma |\neg \downarrow_1 .\pi.\mathbf{true}.\sigma \ \wedge \ P.min(\downarrow_2 .\pi, \sigma) \})\end{array}$$

= { stability lemma 4.9 and $\downarrow_2 .\pi.\mathbf{true} = \mathbf{true}$}

$$\begin{array}{l}(\{\sigma |\downarrow_1 .\pi.\mathbf{true}.\sigma \wedge \ \downarrow_2 .\pi.P.\sigma = tt\}, \\ \{\sigma |\downarrow_1 .\pi.\mathbf{true}.\sigma \wedge \ \downarrow_2 .\pi.\mathbf{true}.\sigma = tt \wedge \ \downarrow_2 .\pi.P.\sigma = tt\}\cup \\ \cup\{\sigma |\neg \downarrow_1 .\pi.\mathbf{true}.\sigma \wedge \ \downarrow_2 .\pi.\mathbf{true}.\sigma = tt \wedge \ \downarrow_2 .\pi.P.\sigma = tt\})\end{array}$$

= { $\downarrow_1 .\pi.\mathbf{true}.\sigma$ and $\downarrow_2 .\pi.P.\sigma$ if and only if $\downarrow_1 .\pi.P.\sigma$ }

$$(\{\sigma |\downarrow_1 .\pi.P.\sigma = tt\}, \{\sigma |\downarrow_2 .\pi.\mathbf{true}.\sigma = tt \wedge \ \downarrow_2 .\pi.P.\sigma = tt\})$$

=

$$(\downarrow_1 .\pi.P, \downarrow_2 .\pi.P)$$

=

$$\pi.P.$$

$\eta^{-1} \circ \eta = id_{ETran^\emptyset}$) Let $m \in ETran^\emptyset$ and $\sigma \in \Sigma$. If $\bot \in m.\sigma$ then $\neg \downarrow_1 .\eta.m.\mathbf{true}.\sigma$. Thus $\eta^{-1}.(\eta.m).\sigma = min(\downarrow_2 .\eta.m, \sigma) \cup \{\bot\}$. Now for a predicate P we have by definition of η that $\downarrow_2 .\eta.m.P.\sigma = tt$ if and only if $P.(m.\sigma \backslash \{\bot\}) = tt$ thus by stability lemma 4.9 $min(\downarrow_2 .\eta.m, \sigma) = m.\sigma \backslash \{\bot\}$. Therefore if $\bot \in m.\sigma$ we have $\eta^{-1}.(\eta.m).\sigma = m.\sigma$.

Otherwise $\bot \notin m.\sigma$ and hence $\downarrow_1 .\eta.m.\mathbf{true}.\sigma = tt$. Thus $\eta^{-1}.(\eta.m).\sigma = min(\downarrow_2 .\eta.m, \sigma)$. Now for a predicate P we have by definition of η that $\downarrow_2 .\eta.m.P.\sigma = tt$ if and only if $P.(m.\sigma \backslash \{\bot\}) = tt$ if and only if $P.m.\sigma = tt$ because $\bot \notin m.\sigma$. Thus by stability lemma 4.9 $min(\downarrow_2 .\eta.m, \sigma) = m.\sigma$ and hence $\bot \notin m.\sigma$ we have $\eta^{-1}.(\eta.m).\sigma = m.\sigma$.

□

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

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

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

Next we give the extension of *wp* and *wlp* to the new set of statements:

Definition 5.1 *(Extension of wp) Let*

$$wp : Stat \rightarrow (Env \rightarrow PTran)$$

for $\xi \in Env$ *be defined by*

$$wp.b \rightarrow .\xi.Q = b \Rightarrow Q$$

$$wp.x.\xi.Q = \xi.x.Q$$

$$wp.v := t.\xi.Q = Q[t/v]$$

$$wp.S_1; S_2.\xi.Q = wp.S_1.\xi.(wp.S_2.\xi.Q)$$

$$wp.S_1 \Box S_2.\xi.Q = wp.S_1.\xi.Q \wedge wp.S_2.\xi.Q$$

$$wp.S_1 \Diamond S_2.\xi.Q = wp.S_1.\xi.Q \wedge (wp.S_1.\xi.\mathbf{false} \Rightarrow wp.S_2.\xi.Q).$$

Definition 5.2 *(Extension of wlp) Let*

$$wlp : Stat \rightarrow (Env \rightarrow PTran)$$

for $\xi \in Env$ *be defined by*

$$wlp.b \rightarrow .\xi.Q = b \Rightarrow Q$$

$$wlp.x.\xi.Q = \xi.x.Q$$

$$wlp.v := t.\xi.Q = Q[t/v]$$

$$wlp.S_1; S_2.\xi.Q = wlp.S_1.\xi.(wlp.S_2.\xi.Q)$$

$$wlp.S_1 \Box S_2.\xi.Q = wlp.S_1.\xi.Q \wedge wlp.S_2.\xi.Q$$

$$wlp.S_1 \Diamond S_2.\xi.Q = wlp.S_1.\xi.Q \wedge (wp.S_1.\xi.\mathbf{false} \Rightarrow wlp.S_2.\xi.Q).$$

Take a fixed declaration $d \in Decl : Pvar \rightarrow Stat$. 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.

We add simultaneous recursion to our language (as opposed to adding for example μ-recursion as it is done in [BvW90, vW90]). Our motivation for having simultaneous recursion is that we developed the original theory for PROLOG programs that have this form of recursion, but we do not expect problems when we add μ-recursion.

First we show how familiar constructions can be defined in a declaration, for example the do-loop **do** S **od** can be defined by the statement $x \Leftarrow (S; x) \Diamond (\mathbf{true} \rightarrow)$, the conditional **if** S **fi** by $y \Leftarrow S \Diamond y$, and *loop* by $z \Leftarrow z$.

Remark: our language has only bounded nondeterminism, and therefore all the ordinals we will encounter below are never bigger than ω. However, if we would add unbounded choice $(\Box : i \in I : S_i)$ then the theory still applies since we have already seen that multiplicativity is equivalent to Σ-multiplicativity.

Define $\phi : Decl \rightarrow (Env \rightarrow Env)$, 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

- $Env_B = (PVar \rightarrow PTran_B)$,
- $Env_D = (PVar \rightarrow PTran_D)$,
- $Env_N = (PVar \rightarrow PTran_N)$,

and ordered pointwise, that is

- $\xi_1 \sqsubseteq_{EB} \xi_2$ *if* $(\forall x \in PVar : \xi_1.x \sqsubseteq_{PB} \xi_2.x)$,
- $\xi_1 \sqsubseteq_{ED} \xi_2$ *if* $(\forall x \in PVar : \xi_1.x \sqsubseteq_{PD} \xi_2.x)$,

- $\xi_1 \sqsubseteq_{EN} \xi_2$ *if* $(\forall x \in PVar : \xi_1.x \sqsubseteq_{PN} \xi_2.x)$.

Theorem 5.3 $(Env_N, \sqsubseteq_{EN})$ *is a complete partial ordering.*

For $k \in \{B, D, N\}$ and $\xi_k \in Env_k$ lift the definition above of ϕ to $\phi_k : Decl \to (Env_k \to Env_k)$ 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 for a fixed declaration d the function $(\phi_B.d)$ is in general not monotone for $\sqsubseteq_{EB}$ while the function $(\phi_D.d)$ is in general not monotone for $\sqsubseteq_{ED}$. Take, for example, for $\sqsubseteq_{EB}$ a declaration that contains

$$x \Leftarrow (x \diamond v := 1),$$

and for $\sqsubseteq_{ED}$ a declaration with

$$x \Leftarrow x; (v = 1 \to),$$

and adapt the examples before Theorem 3.6.

However, using Theorem 4.2 we can define two functions $h_{NB} : Env_N \to Env_B$ and $h_{ND} : Env_N \to Env_D$ by:

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

Both h_{NB} and h_{ND} are onto, monotone and continuous functions. Moreover, for every $\xi \in Env_B$ there is a unique top element in $h_{NB}^{-1}.\xi$; and similarly for every $\xi \in Env_D$ there is a unique top element in $h_{ND}^{-1}.\xi$.

Hence we can apply the Theorem 4.5:

Theorem 5.4 *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

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

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 5.5 *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:*

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

6 Deriving Semantics by Domain Transformations

In the previous section we have given three different semantics for a language including recursion. Now we want to translate this results to semantics based on state transformations. Since we have isomorphisms between the domains of predicate transformers and those of state transformers the semantics can be easily derived, but we want to do it in a more general context using the results of section 4. We give a tecnique to derive a compositional semantics a least fixed point semantics using a domain transformation.

6.1 Compositional Semantics

We start by giving some general definitions (see [GTWW77], [EM85] and reference there). A *signature* $S = (F, r)$ consists of a set $(f \in)F$ of *function names*, and a *rank function* $r : F \to N$, indicating for each function symbol its arity. Function names with arity 0 are called *constants*. In the sequel we will use $f \in S$ instead of $f \in F$ for the signature $S = (F, r)$. Furthermore, we will not use the curry notations for functions with more than one argument. The set of (closed) *terms* $(s, t \in)T(S)$ built from S is defined as

$$t ::= f(t_1, ..., t_{r(f)}).$$

For example, the set of statements *Stat* as defined in section 2 is a set of terms over a signature S_{Stat} in which function names with rank two are ; , $\Box$, and $\Diamond$. No function symbol has rank more than two.

Let V be a set, and define the set of *interpretations* $(I \in)Int_{S,V}$ of a signature $S = (F, r)$ as the set of all functions

$$I : F \to \bigcup_k (V^{(k)} \to V),$$

where $V^{(k)}$ is the k-product of V. An interpretation I induces for every term $t \in T(S)$ a function $t^I : V^{(k)} \to V$ that is given, inductively, by

$$f(t_1, ..., t_{r(f)})^I = I(f)(t_1^I, ..., t_{r(f)}^I).$$

We often write f^I for $I(f)$. We can now give a general definition of semantics:

Definition 6.1 *A semantic function is a function $\mathcal{D} : T(S) \to Dom$ where $T(S)$ is the set of terms over a signature S and Dom is some (structured) set called semantic domain.*

For example, the function *Op* defined in definition 2.3 is a semantic function according to this definition.

Every interpretation $I \in Int_{S,Dom}$ induces a semantic function $\mathcal{D}_I : T(S) \to Dom$ defined by $\mathcal{D}_I(t) = t^I$. This semantics is called compositional because for every term $t = f(u_1, ..., u_{r(f)}) \in T(S)$ we have:

$$\mathcal{D}_I(f(u_1, ..., u_{r(f)}) = f^I(\mathcal{D}_I(u_1), ..., \mathcal{D}_I(u_{r(f)})).$$

We use the following definition of compositionality:

Definition 6.2 *A semantic function $\mathcal{D} : T(S) \to Dom$ is compositional (or denotational) if there exists an interpretation $I \in Int_{S,Dom}$ such that $\mathcal{D} = \mathcal{D}_I$.*

For example, we can interpret the function symbols ; , $\Box$ and $\Diamond$ of the signature S_{Stat} as the following functions from $PTran_N \times PTran_N$ to $PTran_N$. Using this interpretation is not hard to see that the semantics $\mathcal{W}_N : Stat \to (Decl \to PTran_N)$, defined in definition5.5, is compositional.

Definition 6.3 *Let $\pi_1, \pi_2 \in PTran_N$ and $P \in Pred$. Define:*

- $;_N : PTran_N \times PTran_N \to PTran_N$ *by*

$$(\pi_1 ;_N \pi_2).P = (\downarrow_1 .\pi_1.\downarrow_1 .\pi_2.P, \downarrow_2 .\pi_1.\downarrow_2 .\pi_{.2}P),$$

- $\Box_N : PTran_N \times PTran_N \to PTran_N$ *by*

$$(\pi_1 \Box_N \pi_2).P = ((\downarrow_1 .\pi_1 \wedge \downarrow_1 .\pi_2).P, (\downarrow_2 .\pi_1 \wedge \downarrow_2 .\pi_2.P)),$$

- $\Diamond_N : PTran_N \times PTran_N \to PTran_N$ *by*

$$\begin{aligned}(\pi_1 \Diamond_N \pi_2).P = \quad & (\downarrow_1 .\pi_1.P \wedge (\downarrow_1 .\pi_1.\mathbf{false} \Rightarrow \downarrow_1 .\pi_2.P), \\ & \downarrow_2 .\pi_1.P \wedge (\downarrow_1 .\pi_1.\mathbf{false} \Rightarrow \downarrow_2 .\pi_2.P)).\end{aligned}$$

Often compositionality is expressed as a *congruence* for the signature S, that is, an equivalence relation $\equiv \subseteq T(S) \times T(S)$ such that for all $f \in S$ and (closed) term $u_1, .., u_{r(f)}, v_1, ..., v_{r(f)}$

$$(\forall 1 \leq i \leq r(f) : u_i \equiv v_i \;\Rightarrow\; f(u_1, .., u_{r(f)}) \equiv f(v_1, ..., v_{r(f)})).$$

The following lemma is standard, see for example [EM85]:

Lemma 6.4 *Let $\mathcal{D} : T(S) \to Dom$ be a semantic function, and let $\equiv_{\mathcal{D}} \subseteq T(S) \times T(S)$ be defined by $s \equiv_{\mathcal{D}} t \Leftrightarrow \mathcal{D}(s) = \mathcal{D}(t)$. Then $\mathcal{D}$ is compositional if and only if $\equiv_{\mathcal{D}}$ is a congruence.*

If we have a compositional semantics $\mathcal{D} : T(S) \to Dom$ and a function $h : Dom \to Dom'$ we can compose them to obtain another semantics $\mathcal{D}'$ which is compositional if and only if there exist two interpretations $I \in Int_{S,Dom}$ and $I' \in Int_{S,Dom'}$ such that h makes the following diagram commutes for every $f \in S$:

$$\begin{array}{ccc} Dom^{(r(f))} & \xrightarrow{f^I} & Dom \\ \Big\downarrow{\scriptstyle h^{r(f)}} & * & \Big\downarrow{\scriptstyle h} \\ Dom'^{(r(f))} & \xrightarrow[f^{I'}]{} & Dom \end{array}$$

The above fact in itself is simple if the semantics $\mathcal{D}$ and the function h are onto.

Theorem 6.5 *Let $\mathcal{D} : T(S) \to Dom$ a compositional semantic function and $h : Dom \to Dom'$ be a function. Then $\mathcal{D}' = h \circ \mathcal{D} : Stat \to Dom'$ is a compositional semantic function if and only if there exists an interpretation $I \in Int_{S,Dom}$ such that*

1. $\mathcal{D} = \mathcal{D}_I$, *and*

2. $(\forall f \in S : (\forall 1 \leq j \leq r(f) : h(x_j) = h(x'_j) \Rightarrow h(f^I(x_1, ...x_{r(f)})) = h(f^I(x'_1, ...x'_{r(f)}))))$

where $x_j, x'_j \in Dom$ *for all* $1 \leq j \leq r(f)$.

Proof

$\Leftarrow$) Let $f \in S$ and let $u_i, v_i \in T(S)$ such that $u_i \equiv_{\mathcal{D}'} v_i$ for all $1 \leq i \leq r(f)$, that is, $\mathcal{D}'(u_i) = \mathcal{D}'(v_i)$ or, equivalently, $h(\mathcal{D}(u_i)) = h(\mathcal{D}(v_i))$. We have :

$$\mathcal{D}'(f(u_1, ..., u_{r(f)})$$

$= \{$ definition of $\mathcal{D}' \}$

$$h(\mathcal{D}(f(u_1, ..., u_{r(f)})$$

$= \{ \mathcal{D} = \mathcal{D}_I \}$

$$h(f(u_1, ..., u_{r(f)})^I)$$

$= \{$ definition of interpretation $\}$

$$h(f^I(\mathcal{D}(u_1), ..., \mathcal{D}(u_{r(f)})))$$

$= \{$ by hypothesis 2., $f^I(\mathcal{D}(u_1), ..., \mathcal{D}(u_{r(f)})) = f^I(\mathcal{D}(v_1), ..., \mathcal{D}(v_{r(f)}))$ $\}$

$$h(f^I(\mathcal{D}(v_1), ..., \mathcal{D}(v_{r(f)})))$$

$= \{$ by hypothesis 1., $\mathcal{D} = \mathcal{D}_I$ $\}$

$$h(\mathcal{D}(f(v_1, ..., v_{r(f)})$$

$= \{$ definition of $\mathcal{D}'$ $\}$

$$\mathcal{D}'(f(v_1, ..., v_{r(f)}).$$

Hence $\equiv_{\mathcal{D}'}$ is a congruence or, equivalently, $\mathcal{D}'$ is compositional.

$\Rightarrow$) Let $\mathcal{D}' = h \circ \mathcal{D}$ be a compositional semantic function, say $\mathcal{D}' = \mathcal{D}_{I'}$ for some interpretation $I' \in Int_{S,Dom'}$. Because $\mathcal{D}$ is also compositional we have $\mathcal{D} = \mathcal{D}_I$ for some interpretation $I \in Int_{S,Dom}$. We have to prove that there exists an interpretation $\hat{I} \in Int_{S,Dom}$ such that 1. and 2. hold. For every $f \in S$, define

$$\hat{I}(f(x_1, ..., x_{r(f)})) = \begin{cases} f^I(x_1, ..., x_{r(f)}) & \text{if } (\forall 1 \leq i \leq r(f) : (\exists t_i \in T(S) : \mathcal{D}(t_i) = x_i)) \\ x & \text{otherwise,} \end{cases}$$

for an arbitrary $x \in h^{-1}(f^{I'}(h(x_1), ..., h(x_{r(f)})))$ if it is non-empty, otherwise for a fixed $x \in Dom$. By induction we can prove 1.:

$$\mathcal{D}(f(t_1, ..., t_{r(f)}))$$

$= \{ \mathcal{D} = \mathcal{D}_I \}$

$$f^I(\mathcal{D}(t_1), ..., \mathcal{D}(t_{r(f)}))$$

$= \{$ definition of $\hat{I} \}$

$$f^{\hat{I}}(\mathcal{D}(t_1), ..., \mathcal{D}(t_{r(f)}))$$

$= \{$ induction hypothesis $\}$

$$\mathcal{D}_{\hat{I}}(f(t_1, ..., t_{r(f)})).$$

Now we are going to prove 2.. Let $h(x_i) = h(x_i')$ for all $1 \leq i \leq r(f)$. Note that if $h^{-1}(f^{I'}(h(x_1), ..., h(x_{r(f)})))$ is empty then also $h^{-1}(f^{I'}(h(x_1'), ..., h(x_{r(f)}')))$ is empty because $h(x_i) = h(x_i')$ for all $1 \leq i \leq r(f)$. Moreover, if not for each $1 \leq i \leq r(f)$ there exists a $t_i \in T(S)$ such that $\mathcal{D}(t_i) = x_i$, then also not for each $1 \leq i \leq r(f)$ there exists a $t_i \in T(S)$ such that $\mathcal{D}(t_i) = x_i'$, because if we assume that $(\forall 1 \leq i \leq r(f) : (\exists t_i \in T(S) : \mathcal{D}(t_i) = x_i')$, then we get a contradction:

$$h(f^I(\mathcal{D}(t_1), ..., \mathcal{D}(t_{r(f)})))$$

$=$

$$h(\mathcal{D}(f(t_1, ..., t_{r(f)})))$$

$=$

$$\mathcal{D}'(f(t_1, ..., t_{r(f)}))$$

$=$

$$f^{I'}(h(\mathcal{D}(t_1)), ..., h(\mathcal{D}(t_{r(f)})))$$

$=$

$$f^{I'}(h(x_1), ..., h(x_{r(f)})).$$

This contradicts that $h^{-1}(f^{I'}(h(x_1), ..., h(x_{r(f)})))$ is empty. Therefore in this case we have:

$$h(f^{\hat{I}}(x_1, ..., x_{r(f)})) = h(x) = h(f^{\hat{I}}(x_1', ..., x_{r(f)}')),$$

where $x \in Dom$. There remain two cases to prove. We will show only $h(f^{\hat{I}}(x_1, ..., x_{r(f)})) = f^{I'}((h(x_1'), ..., h(x_{r(f)}'))$ because the proof of $f^{I'}(h(x_1'), ..., h(x_{r(f)}')) = h(f^{\hat{I}}(x_1', ..., x_{r(f)}'))$ is similar.

If $(\forall 1 \leq i \leq r(f) : (\exists t_i \in T(S) : \mathcal{D}(t_i) = x_i))$ then we have:

$$h(f^{\hat{I}}(x_1, ..., x_{r(f)}))$$

$= \{$ definition of $\hat{I}$ $\}$

$h(f^I(\mathcal{D}(t_1), ..., \mathcal{D}(t_{r(f)})))$

$= \{ \mathcal{D} = \mathcal{D}_I \}$

$h(\mathcal{D}(f(t_1, ..., t_{r(f)})))$

$= \{\mathcal{D}' = h \circ \mathcal{D} \}$

$\mathcal{D}'(f(t_1, ..., t_{r(f)}))$

$= \{ \mathcal{D}' = \mathcal{D}_{I'} \}$

$f^{I'}(\mathcal{D}'(t_1), ..., \mathcal{D}'(t_{r(f)}))$

$= \{ \mathcal{D}' = h \circ \mathcal{D} \}$

$f^{I'}(h(\mathcal{D}(t_1)), ..., h(\mathcal{D}'(t_{r(f)})))$

$= \{ \mathcal{D}(t_i) = x_i$ for all $1 \leq i \leq r(f) \}$

$f^{I'}(h(x_1), ..., h(x_{r(f)}))$

$= \{ h(x_i) = h(x_i')$ for all $1 \leq i \leq r(f) \}$

$f^{I'}(h(x_1'), ..., h(x_{r(f)}'))$.

If $\neg(\forall 1 \leq i \leq r(f) : (\exists t_i \in T(S) : \mathcal{D}(t_i) = x_i))$ and $h^{-1}(f^{I'}(h(x_1), ..., h(x_{r(f)}))) \neq \emptyset$, we have:

$h(f^{\hat{I}}(x_1, ..., x_{r(f)}))$

$= \{$ definition of $\hat{I}$ $\}$

$h(x)$

$= \{ x \in h^{-1}(f^{I'}(h(x_1), ..., h(x_{r(f)}))) \}$

$f^{I'}(h(x_1), ..., h(x_{r(f)}))$

$= \{ h(x_i) = h(x_i')$ for all $1 \leq i \leq r(f) \}$

$f^{I'}(h(x_1'), ..., h(x_{r(f)}'))$.

In a similar way one can prove $f^{I'}(h(x_1'), ..., h(x_{r(f)}')) = h(f^{\hat{I}}(x_1', .., x_{r(f)}'))$. Finally we obtain

$h(f^{\hat{I}}(x_1, ..., x_{r(f)})) = h(f^{\hat{I}}(x_1', .., x_{r(f)}'))$.

□

Remark: if h is injective then the theorem 6.5 holds trivially.

Applying theorem 6.5 we have that the semantic function $\mathcal{W}_D : Stat \to (Decl \to PTran_D)$, defined in definition 5.5, is compositional because $\mathcal{W}_N$ is compositional, and the following fact holds:

Theorem 6.6 *Let π_1, π_2, π_1', and $\pi_2' \in PTran_N$ and let $\tilde{op} \in \{;_N, \Box_N, \Diamond_N\}$. Then*

$$\downarrow_1 .\pi_1 = \downarrow_1 .\pi_1' \wedge \downarrow_1 .\pi_2 = \downarrow_1 .\pi_2' \Rightarrow \downarrow_1 .(\pi_1 \ \tilde{op} \ \pi_2) = \downarrow_1 .(\pi_1' \ \tilde{op} \ \pi_2')$$

In a similar way we can prove $\mathcal{W}_B : Stat \to (Decl \to PTran_B)$ is a compositional semantic function. Hence we can define an interpretation in $PTran_D$ and $PTran_B$ of the function symbols $;$, $\Box$ and $\Diamond$ of the signature $Stat$ of the previous section, starting from their interpretation in $PTran_N$. Moreover, because the function ω, γ, η defined in the previous sections are injective theorem 6.5 holds, and hence we can give an interpretation of $;$, $\Box$ and $\Diamond$ on $ETran^{\emptyset}, STran^{\delta}$, and $STran^{\emptyset}$ starting from the interpretation in $PTran_N$. In the next definition we give this interpretation on $ETran^{\emptyset}$ and then we prove its correctness with respect to $PTran_N$:

Definition 6.7 *Let $m_1, m_2 \in ETran^{\emptyset}$ and $\sigma \in \Sigma$, define*

- $;_{ETran^{\emptyset}} : ETran^{\emptyset} \times ETran^{\emptyset} \to ETran^{\emptyset}$ *by:*

$$(m_1 \ ;_{ETran^{\emptyset}} \ m_2).\sigma = \begin{cases} \perp & \text{if } m_1.\sigma = \perp \\ \emptyset & \text{if } m_1.\sigma = \emptyset \\ \bigcup m_2.(m_1.\sigma \backslash \{\perp\}) \cup & \\ \quad \cup \{\perp \mid \perp \in m_1.\sigma\} & \text{otherwise.} \end{cases}$$

- $\Box_{ETran^{\emptyset}} : ETran^{\emptyset} \times ETran^{\emptyset} \to ETran^{\emptyset}$ *is defined by:*

$$(m_1 \ \Box_{ETran^{\emptyset}} \ m_2).\sigma = m_1.\sigma \cup m_2.\sigma.$$

- $\Diamond_{ETran^{\emptyset}} : ETran^{\emptyset} \times ETran^{\emptyset} \to ETran^{\emptyset}$ *is defined by:*

$$(m_1 \ \Diamond_{ETran^{\emptyset}} \ m_2).\sigma = \begin{cases} m_2.\sigma & \text{if } m_1.\sigma = \emptyset \\ m_1.\sigma & \text{otherwise.} \end{cases}$$

Theorem 6.8 *Let $m_1, m_2 \in ETran^{\emptyset}$. Then*

$$\eta.(m_1 \ ;_{ETran^{\emptyset}} \ m_2) = (\eta \circ m_1 \ ;_N \ \eta \circ m_2)$$
$$\eta.(m_1 \ \Box_{ETran^{\emptyset}} \ m_2) = (\eta \circ m_1 \ \Box_N \ \eta \circ m_2)$$
$$\eta.(m_1 \ \Diamond_{ETran^{\emptyset}} \ m_2) = (\eta \circ m_1 \ \Diamond_N \ \eta \circ m_2)$$

Proof

- $\eta.(m_1 \ ;_{ETran^{\emptyset}} \ m_2).P$

  $=$ { definition of η }

$$(\{\sigma|P.(m_1 \ ;_{ETran^\emptyset}\ m_2).\sigma\},\ \{\sigma|P.((m_1 \ ;_{ETran^\emptyset}\ m_2).\sigma\backslash\{\bot\})\})$$

= { definition of $m_1 \ ;_{ETran^\emptyset}\ m_2$. (We only give the case that $m_1 \neq \bot$ and $m_1 \neq \emptyset$) }

$$(\{\sigma|P.(\bigcup m_2.(m_1.\sigma\backslash\{\bot\}) \cup \{\bot \mid \bot \in m_1.\sigma\})\},\ \{\sigma|P.((\bigcup m_2.(m_1.\sigma\backslash\{\bot\}) \cup \{\bot \mid \bot \in m_1.\sigma\})\backslash\{\bot\})\})$$

=

$$(\{\sigma|\ \bot \notin m_1.\sigma \ \wedge\ P.(\bigcup m_2.(m_1.\sigma\backslash\bot)\},\ \{\sigma|P.(\bigcup m_2.(m_1.\sigma\backslash\{\bot\})\backslash\{\bot\})\})$$

=

$$(\{\sigma|\ \bot \notin m_1.\sigma \ \wedge\ (\forall\sigma' \in m_1.\sigma : P.m_2.\sigma')\},\ \{\sigma|(\forall\sigma' \in (m_1.\sigma\backslash\{\bot\}) : P.(m_2.\sigma'\backslash\{\bot\}))\})$$

=

$$(\{\sigma|\ \bot \notin m_1.\sigma \ \wedge\ m_1.\sigma \subseteq \{\sigma'|P.m_2.\sigma'\}\},\ \{\sigma|(m_1.\sigma\backslash\{\bot\}) \subseteq \{\sigma'|P.(m_2.\sigma'\backslash\{\bot\})\}\})$$

= { definition of η and considering $m_1.\sigma$ as a predicate }

$$(\{\sigma|(\downarrow_1 .\eta.m_2.P).m_1.\sigma\}, \{\sigma|(\downarrow_1 .\eta.m_2.P).(m_1.\sigma\backslash\{\bot\})\})$$

= { definition of η }

$$(\downarrow_1 .\eta.m_1.(\downarrow_1 .\eta.m_2.P), \downarrow_2 .\eta.m_1.(\downarrow_2 .\eta.m_2.P))$$

= { definition of $;_N$ }

$$(\eta.m_1 \ ;_N\ \eta.m_2).P$$

- $\eta.(m_1 \ \Box_{ETran^\emptyset}\ m_2).P$

= { definition of η }

$$(\{\sigma|P.(m_1 \ \Box_{ETran^\emptyset}\ m_2).\sigma\},\ \{\sigma|P.((m_1 \ \Box_{ETran^\emptyset}\ m_2).\sigma\backslash\{\bot\})\})$$

= { definition of $m_1 \ \Box_{ETran^\emptyset}\ m_2$ }

$$(\{\sigma|P.(m_1.\sigma \cup m_2.\sigma)\},\ \{\sigma|P.((m_1.\sigma \cup m_2.\sigma)\backslash\{\bot\})\})$$

=

$$(\{\sigma|P.m_1.\sigma \wedge P.m_2.\sigma\},\ \{\sigma|P.((m_1.\sigma\backslash\{\bot\}) \wedge P.(m_2.\sigma\backslash\{\bot\})\})$$

=

$$(\downarrow_1 .\eta.m_1.P \wedge \downarrow_1 .\eta.m_2.P, \downarrow_2 .\eta.m_1.P \wedge \downarrow_2 .\eta.m_2.P)$$

$=$

$$(\eta.m_1 \ \Box_N \ \eta.m_2).P$$

- $\eta.(m_1 \ \Diamond_{ETran^\emptyset} \ m_2).P$

  $=$ { definition of η }

  $$(\{\sigma | P.(m_1 \ \Diamond_{ETran^\emptyset} \ m_2).\sigma\},$$
  $$\{\sigma | P.((m_1 \ \Diamond_{ETran^\emptyset} \ m_2).\sigma \backslash \{\bot\})\})$$

  $=$ { definition of $m_1 \ \Diamond_{ETran^\emptyset} \ m_2$ }

  $$(\{\sigma | P.m_2.\sigma \ \wedge \ m_1.\sigma = \emptyset\} \cup \{\sigma | P.m_2.\sigma \ \wedge \ m_1.\sigma \neq \emptyset\},$$
  $$\{\sigma | P.(m_2.\sigma \backslash \{\bot\}) \ \wedge \ m_1.\sigma = \emptyset\} \cup \{\sigma | P.(m_1.\sigma \backslash \{\bot\}) \ \wedge \ m_1.\sigma \neq \emptyset\})$$

  $=$ { definition of η }

  $$((\downarrow_1 .\eta.m_2.P \wedge \downarrow_1 .\eta.m_1.\mathbf{false}) \vee (\downarrow_1 .\eta.m_1.P \ \wedge \ \neg \downarrow_1 .\eta.m_1.\mathbf{false}),$$
  $$(\downarrow_2 .\eta.m_2.P \wedge \downarrow_1 .\eta.m_1.\mathbf{false}) \vee (\downarrow_2 .\eta.m_1.P \ \wedge \ \neg \downarrow_1 .\eta.m_1.\mathbf{false}))$$

  $=$

  $$(\downarrow_1 .\eta.m_1.P \ \wedge \ (\downarrow_1 .\eta.m_1.\mathbf{false} \Rightarrow \downarrow_1 .\eta.m_2.P),$$
  $$\downarrow_2 .\eta.m_1.P \ \wedge \ (\downarrow_1 .\eta.m_1.\mathbf{false} \Rightarrow \downarrow_2 .\eta.m_2.P))$$

  $=$ { definition of $\Diamond_N$ }

  $$(\eta.m_1 \ \Diamond_N \ \eta.m_2).P$$

□

Next we give the definition of the interpretation of $;, \Box, \Diamond$ in $STran^\delta$, and next we prove its correctness with respect to $PTran_D$ and $ETran^\emptyset$:

Definition 6.9 *Let $m_1, m_2 \in STran^\delta$ and $\sigma \in \Sigma$, define*

- $;_{STran^\delta} : STran^\delta \times STran^\delta \to STran^\delta$ *by:*

$$(m_1 \ ;_{STran^\delta} \ m_2).\sigma = \begin{cases} \Sigma_\bot & \text{if } m_1.\sigma = \Sigma_\bot \\ \delta & \text{if } m_1.\sigma = \delta \\ \bigcup m_2.m_1.\sigma & \text{otherwise.} \end{cases}$$

- $\Box_{STran^\delta} : STran^\delta \times STran^\delta \to STran^\delta$ *by:*

$$(m_1 \ \Box_{STran^\delta} \ m_2).\sigma = \begin{cases} m_2.\sigma & \text{if } m_1.\sigma = \delta \\ m_1.\sigma & \text{if } m_2.\sigma = \delta \\ m_1.\sigma \cup m_2.\sigma & \text{otherwise.} \end{cases}$$

- $\diamond_{STran^\delta} : STran^\delta \times STran^\delta \to STran^\delta$ *by:*

$$(m_1 \diamond_{STran^\delta} m_2).\sigma = \begin{cases} m_2.\sigma & \text{if } m_1.\sigma = \delta \\ m_1.\sigma & \text{otherwise.} \end{cases}$$

Theorem 6.10 *Let $m_1, m_2 \in STran^\delta$. We have:*

1. $\gamma.(m_1 \;;_{STran^\delta}\; m_2) = (\gamma \circ m_1 \;;_D\; \gamma \circ m_2)$
   $\gamma.(m_1 \;\Box_{STran^\delta}\; m_2) = (\gamma \circ m_1 \;\Box_D\; \gamma \circ m_2)$
   $\gamma.(m_1 \;\diamond_{STran^\delta}\; m_2) = (\gamma \circ m_1 \;\diamond_D\; \gamma \circ m_2)$
2. $e_\Sigma.(m_1 \;;_{ETran^\emptyset}\; m_2) = (e_\Sigma \circ m_1 \;;_{STran^\delta}\; e_\Sigma \circ m_2)$
   $e_\Sigma.(m_1 \;\Box_{ETran^\emptyset}\; m_2) = (e_\Sigma \circ m_1 \;\Box_{STran^\delta}\; e_\Sigma \circ m_2)$
   $e_\Sigma.(m_1 \;\diamond_{ETran^\emptyset}\; m_2) = (e_\Sigma \circ m_1 \;\diamond_{STran^\delta}\; e_\Sigma \circ m_2)$

Finally we give the definition of the interpretation of $;, \Box, \diamond$ in $STran^\emptyset$, and next we prove its correctness with respect to $PTran_B$ and $STran^\delta$:

Definition 6.11 *Let $m_1, m_2 \in STran^\emptyset$ and $\sigma \in \Sigma$, define*

- $;_{STran^\emptyset} : STran^\emptyset \times STran^\emptyset \to STran^\emptyset$ *by:*

$$(m_1 \;;_{STran^\emptyset}\; m_2).\sigma = \begin{cases} \Sigma_\perp & \text{if } m_1.\sigma = \Sigma_\perp \\ \emptyset & \text{if } m_1.\sigma = \emptyset \\ \bigcup m_2.m_1.\sigma & \text{otherwise.} \end{cases}$$

- $\Box_{STran^\emptyset} : STran^\emptyset \times STran^\emptyset \to STran^\emptyset$ *by:*

$$(m_1 \;\Box_{STran^\emptyset}\; m_2).\sigma = m_1.\sigma \cup m_2.\sigma.$$

- $\diamond_{STran^\emptyset} : STran^\emptyset \times STran^\emptyset \to STran^\emptyset$ *by:*

$$(m_1 \;\diamond_{STran^\emptyset}\; m_2).\sigma = \begin{cases} m_2.\sigma & \text{if } m_1.\sigma = \emptyset \\ m_1.\sigma & \text{otherwise.} \end{cases}$$

Theorem 6.12 *Let $m_1, m_2 \in STran^\emptyset$. Then*

1. $\omega.(m_1 \;;_{STran^\emptyset}\; m_2) = (\omega \circ m_1 \;;_B\; \omega \circ m_2)$
   $\omega.(m_1 \;\Box_{STran^\emptyset}\; m_2) = (\omega \circ m_1 \;\Box_B\; \omega \circ m_2)$
   $\omega.(m_1 \;\diamond_{STran^\emptyset}\; m_2) = (\omega \circ m_1 \;\diamond_B\; \omega \circ m_2)$
2. $d_\Sigma.(m_1 \;;_{STran^\delta}\; m_2) = (d_\Sigma \circ m_1 \;;_{STran^\emptyset}\; d_\Sigma \circ m_2)$
   $d_\Sigma.(m_1 \;\Box_{STran^\delta}\; m_2) = (d_\Sigma \circ m_1 \;\Box_{STran^\emptyset}\; d_\Sigma \circ m_2)$
   $d_\Sigma.(m_1 \;\diamond_{STran^\delta}\; m_2) = (d_\Sigma \circ m_1 \;\diamond_{STran^\emptyset}\; d_\Sigma \circ m_2)$

6.2 Fixed point Semantics

Recursion can be added to a language by means of a set of constants $(x \in) PVar$, called *procedure variables* that is added to a signature S (as we have done in the previous section). Let $S_{rec} = S \cup Pvar$ be the new signature, the meaning of procedure variables is then given by means of a fixed point of a function associated to a *declaration* $d : Pvar \to Stat$. Given a semantic function $\mathcal{D} : T(S_{rec}) \to Dom$ we denote by $\mathcal{D}_0 : T(S) \to Dom$ its restriction to terms without procedure variables. The set of *environments* is given by $(\xi \in) Env : Pvar \to Dom$. Every compositional semantics $\mathcal{D} : T(S) \to Dom$ induces a compositional semantics $\hat{\mathcal{D}} : T(S_{rec}) \to (Env \to Dom)$ using the environments; it is defined by

$$\begin{array}{ll} \hat{\mathcal{D}}(x)(\xi) = \xi(x) & \text{for each } x \in PVar \\ \hat{\mathcal{D}}(f(t_1, ..., t_{r(f)})(\xi) = f^I(\hat{\mathcal{D}}(t_1)(\xi), ..., \hat{\mathcal{D}}(t_{r(f)})(\xi)) & \text{for each } f \in T(S) \text{ and } t_i \in T(S_{rec}), \end{array}$$

where $I \in Int_{S,Dom}$ such that $\mathcal{D} = \mathcal{D}_I$. Using this semantics we can define the function $\phi_{\hat{\mathcal{D}}} : Env \to Env$ by

$$\phi_{\hat{\mathcal{D}}}(\xi)(x) = \hat{\mathcal{D}}(d(x))(\xi).$$

Let $fix.\phi_{\hat{\mathcal{D}}}$ denote the set (possibly empty) of its fixed points. Elements of this set are used for giving meaning to procedure variables.

Definition 6.13 *A semantic function $\mathcal{D} : T(S_{rec}) \to Dom$ is a fixed point semantics if the semantics $\mathcal{D}_0 : T(S) \to Dom$ is compositional and $\mathcal{D}(t) = \hat{\mathcal{D}}_0(t)(\hat{\xi})$, where $\hat{\xi} \in fix.\phi_{\hat{\mathcal{D}}_0}$. Furthermore, $\mathcal{D} : T(S_{rec}) \to Dom$ is a least fixed point semantics if Dom is a partial order and $\hat{\xi} = \mu.\phi_{\hat{\mathcal{D}}_0}$.*

For example, the semantic function $\mathcal{W}_N$ is a least fixed point semantics because it is compositional and, for each $x \in Pvar$, we have

$$\mathcal{W}_N.x.d$$

$=$

$$(wp.x.\downarrow_1.(\mu.(\phi_N.d)), wlp.x.\downarrow_2.(\mu.(\phi_N.d)))$$

$=$

$$(\downarrow_1.(\mu.(\phi_N.d)).x, \downarrow_2.(\mu.(\phi_N.d)).x)$$

$=$

$$(\mu.\phi_N.d).x,$$

where $\phi_N.d.\xi.x = (wp.(d.x).(\downarrow_1.\xi), wlp.(d.x).(\downarrow_2.\xi))$ (as defined in the previous section).

The following property is useful to transfer the property of being a fixed point semantics to another semantics using a domain transformation:

Lemma 6.14 *For a signature S and a set of constants $PVar$ let $S_{rec} = S \cup PVar$ be a new signature. Let $\mathcal{D} : T(S) \to Dom$ be a compositional semantics and $h : Dom \to Dom'$ be a function such that $\mathcal{D}' = h \circ \mathcal{D} : T(S) \to Dom'$ is a compositional semantics. Then for every $t \in S$ and $\xi \in Env$ we have*

$$\hat{\mathcal{D}}'(t)(h \circ \xi) = h(\hat{\mathcal{D}}(t)(\xi))$$

where $\hat{\mathcal{D}}' : T(S_{rec}) \to (Env' \to Dom')$ and $\hat{\mathcal{D}} : T(S_{rec}) \to (Env \to Dom)$ are the semantics induced by $\hat{\mathcal{D}}'$ and $\hat{\mathcal{D}}$, respectively, and $Env' = Pvar \to Dom'$.

Proof By theorem 6.5 there are two interpretations $I \in Int_{S,Dom}$ and $I' \in Int_{S,Dom'}$ such that for every $f \in S$ the following diagram commutes:

$$\begin{array}{ccc} Dom^{(r(f))} & \xrightarrow{f^I} & Dom \\ \Big\downarrow{\scriptstyle h^{r(f)}} & * & \Big\downarrow{\scriptstyle h} \\ Dom'^{(r(f))} & \xrightarrow[f^{I'}]{} & Dom. \end{array}$$

We prove the lemma by inductions on the rank of S_{rec}:

$r(f) = 0$) If $f \in PVar$ then

$$\hat{\mathcal{D}}'(f)(h \circ \xi) = h(\xi(f)) = h(\hat{\mathcal{D}}(f)(\xi).$$

If, instead, $f \in S$ then

$$\hat{\mathcal{D}}'(f)(h \circ \xi) = \mathcal{D}'(f) = h(\mathcal{D}(f)) = h(\hat{\mathcal{D}}(f)(\xi)).$$

$r(f) > 0$) Then we have:

$$\hat{\mathcal{D}}'(f(u_1, ..., u_{r(f)})(h \circ \xi)$$

$= \{ \mathcal{D}' = \mathcal{D}_{I'} \}$

$$f^{I'}(\hat{\mathcal{D}}'(u_1)(h \circ \xi), ..., \hat{\mathcal{D}}'(u_{r(f)})(h \circ \xi))$$

$= \{$ induction hypothesis $\}$

$$f^{I'}(h(\hat{\mathcal{D}}(u_1)(\xi)), ..., h(\hat{\mathcal{D}}(u_{r(f)})(\xi)))$$

$= \{$ commutativity of the diagram above $\}$

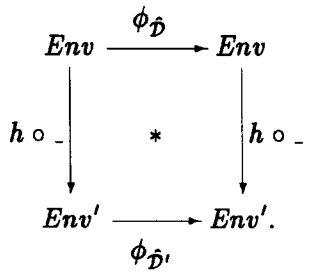
$$h(f^I(\hat{\mathcal{D}}(u_1)(\xi), ..., \hat{\mathcal{D}}(u_{r(f)})(\xi)))$$

$= \{ \mathcal{D} = \mathcal{D}_I \}$

$$h(\hat{\mathcal{D}}(f(u_1, ..., u_{r(f)})(\xi).$$

□

As a consequence of this theorem, we have that the following diagram commutes, for $\phi_{\hat{\mathcal{D}}}(\xi)(x) = \hat{\mathcal{D}}(d(x))(\xi)$ and $\phi_{\hat{\mathcal{D}}'}(\xi)(x) = \hat{\mathcal{D}}'(d(x))(\xi)$:

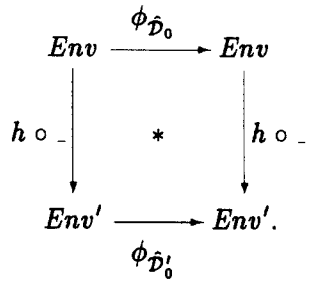


Indeed, we have

$$\phi_{\hat{\mathcal{D}}'}(h \circ \xi)(x) = \hat{\mathcal{D}}'(d(x))((h \circ \xi) = h(\hat{\mathcal{D}}(d(x))(\xi)) = h(\phi_{\hat{\mathcal{D}}}(\xi)(x))$$

Finally, applying the transfer lemmas 4.4, 4.5, and 4.6 we can prove the following theorem that can be seen as a generalization of the Mezei-Wright theorem on ω-complete algebras [Wec92]:

Theorem 6.15 *Let S_{rec} be a signature union of a signature S and a set of constant PVar. Let, also, $\mathcal{D} : T(S_{rec}) \to Dom$ be a least fixed points semantics, $\mathcal{D}_0 : T(S) \to Dom$ be its restriction to terms without procedure variables and $h : Dom \to Dom'$ be a function such that $\mathcal{D}'_0 = h \circ \mathcal{D}_0 : T(S) \to Dom'$ is a compositional semantics. If Dom is a complete partial order and if $\phi_{\hat{\mathcal{D}}_0}$ is monotone and if the commuting diagram*



satisfies one of the following points

1. *h is strict and continuos, and $\phi_{\hat{\mathcal{D}}'_0}$ is monotone,*
2. *h is onto, continuous and for all the $y \in Dom'$ either the lower fringe of $h^{-1}.y$ exists and it is finite, or every antichain of $h^{-1}.y$ is finite,*
3. *h is onto, monotone and for all the $y \in Dom'$ the upper fringe of $h^{-1}.y$ exists and it is finite,*

then also $\mathcal{D}' = h \circ \mathcal{D} : T(S_{rec}) \to Dom'$ is a least fixed point semantics.

If the function $h : Dom \longrightarrow Dom'$ is an isomorphism, then theorem 6.15 holds trivially, thus we can derive easily three least fixed point semantics based on state transformations from the weakest precondition semantics of definition 5.5:

Definition 6.16 *Let $S \in Stat$ and $d \in Decl$. We define the following semantics functions based on state transformations*

- $\mathcal{D}_N = \eta^{-1} \circ \mathcal{W}_N : Stat \to (Decl \to ETran^{\emptyset})$,
- $\mathcal{D}_D = \gamma^{-1} \circ \mathcal{W}_D : Stat \to (Decl \to STran^{\delta})$ *and*
- $\mathcal{D}_B = \omega^{-1} \circ \mathcal{W}_B : Stat \to (Decl \to STran^{\emptyset})$.

7 Conclusions and future work

At least four different, but in some way related, topics have been treated in this paper:

1. We proposed an extension of the Dijkstra's Weakest Precondition Calculus in order to treat recursion in a fully compositional way with respect to three different orders: a refinement order as introduced in [Bac78], a new refinement order that respects deadlock, and an approximation order as introduced in [Nel87].
2. We showed that (under certain circumstances), least fixed points of functions (even non-monotone) between partial orders exist and that they can be obtained by iteration from the least element.
3. We gave three isomorphisms between domains of predicate transformers and three domain of state transformations: two differents versions of the (flat) Smyth state transformers and the Egli-Milner state transformers.
4. We showed that a semantic function can be derived from another semantic function by a domain transformation, preserving proprieties like compositionality and least fixed points.

We like to consider further extensions of the language, like arbitrary parallelism and angelic choice.

Also, we think that applications of the transfer techniques used in the present paper for obtaining the least fixed points of function, can be successfull in the area of logic programming and concurrent systems.

Finally, further investigations are needed on the relationships between predicate transformers and state transformers, considering also non-trivial information systems.

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 Nicoletta Sabadini, Giancarlo Mauri, 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.

[AP81] K. R. Apt and G. Plotkin. A Cook's tour of countable nondeterminism. In S. Evens and O. Kariv, editors, *Proc. 8th ICALP, Akko*, number 115 in Lecture Notes in Computer Science, 1981.

[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

[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.

[dR76] W.P. de Roever. Dijkstra's predicate transformer, non-determinism, recursion, and terminations. In *Mathematical foundations of computer science*, volume 45 of *Lecture Notes in Computer Science*, pages 472–481. Springer-Verlag, Berlin, 1976.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1*, volume 6 of *EATCS monographs.* Springer-Verlag, 1985.

[GTWW77] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *Journ. ACM*, 24:68–95, 1977.

[Heh79] E.C.R. Hehner. do consider od: a contribution to programming calculus. *Acta informatica*, 11:287–304, 1979.

[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.

[Mey85] J.-J.Ch. Meyer. *Programming Calculi Based on Fixed Point Transformations: Semantics and Applications.* PhD thesis, Vrije Universiteit, Amsterdam, 1985.

[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.

[vW90] J. von Wright. *A Lattice-theoretical Basis for Program Refinement.* PhD thesis, Abo Akademi, 1990.

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

[Wec92] E. Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *EATCS monographs.* Springer-Verlag, 1992.

Appendix

In this appendix we give a proof of Theorem 2.5:

Proof Structural induction on $S \in Stat$:

- $v := t$) $\quad \{\sigma | P.Op.(v := t).\sigma\}$

  $= \{$ definition of Op, P. $\perp =$ **false**$\}$

$$\{\sigma|P.\{\sigma'|\langle v := t, \sigma\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \wedge \ (v := t, \sigma)\downarrow\}\}$$

$= \{$definition of $\longrightarrow \}$

$$\{\sigma|P.\sigma[t.\sigma/v]\}$$

$=$ { because $P[t/v].\sigma = P.\sigma[t.\sigma/v]$ }

$$\{\sigma|P[t/v].\sigma\}$$

$=$

$$P[t/v]$$

$=$

$$wp.v := t.P.$$

- $b \rightarrow$) $\{\sigma|P.Op.(b \rightarrow).\sigma\}$

$=$ { definition of Op, $P.\ \perp =$ **false** }

$$\{\sigma|P.\{\sigma'|\langle b \rightarrow, \sigma\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \wedge \ (b \rightarrow, \sigma)\downarrow\}\} =$$

$=$ { definition of $\longrightarrow$ }

$$\{\sigma|(b.\sigma \wedge P.\sigma) \vee (\neg b.\sigma \wedge P.\emptyset)\}$$

$=$ { because $P.\emptyset =$ **true** }

$$\{\sigma|(b.\sigma \wedge P.\sigma) \vee \neg b.\sigma\}$$

$=$

$$\{\sigma|b.\sigma \Rightarrow P.\sigma\}$$

$=$

$$b \Rightarrow P$$

$=$

$$wp.b \rightarrow .P$$

- *loop*) $\{\sigma|P.Op.loop.\sigma\}$

$=$ {definition of Op }

$$\{\sigma|P.\Sigma_\perp\}$$

$=$ { $P.\ \perp =$ **false** }

false

=

$wp.loop.P.$

- $S_1; S_2$) $\{\sigma | P.Op.S_1; S_2.\sigma)\}$

= { definition of Op, $P.\ \bot =$ **false** }

$\{\sigma | P.\{\sigma' | \langle S_1; S_2, \sigma\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \wedge\ (S_1; S_2, \sigma)\downarrow\}\}$

= { definition of $\longrightarrow$ }

$\{\sigma | P.\{\sigma' | (\langle S_1, \sigma\rangle \longrightarrow^* \langle E, \sigma''\rangle \ \wedge\ (S_1, \sigma)\downarrow) \wedge (\langle S_2, \sigma''\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \wedge\ (S_2, \sigma'')\downarrow)\}\}$

= { definition of Op }

$\{\sigma | P.\{\sigma' | \sigma'' \in Op.S_1.\sigma \ \wedge\ \langle S_2, \sigma''\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \wedge\ (S_2, \sigma'')\downarrow\}\}$

=

$\{\sigma | (\forall \sigma'' \in Op.S_1.\sigma \ :\ P.\{\sigma' | \langle S_2, \sigma''\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \wedge\ (S_2, \sigma'')\downarrow\})\}$

= { definition of Op }

$\{\sigma | (\forall \sigma'' \in Op.S_1.\sigma \ :\ P.Op.S_2.\sigma'')\}$

=

$\{\sigma | \{\sigma'' | P.Op.S_2.\sigma''\}.Op.S_1.\sigma\}$

= { induction }

$wp.S_1.\{\sigma'' | P.Op.S_2.\sigma''\}$

{ induction }

$wp.S_1.(wp.S_2.P).$

- $S_1 \Box S_2$) $\{\sigma | P.Op.S_1 \Box S_2.\sigma\}$

= { definition of Op, $P.\ \bot =$ **false** }

$\{\sigma | P.\{\sigma' | \langle S_1 \Box S_2, \sigma\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \wedge\ (S_1 \Box S_2, \sigma)\downarrow\}\}$

= { definition of $\longrightarrow$ }

$\{\sigma | P.\{\sigma' | \langle S_1, \sigma\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \vee\ \langle S_2, \sigma\rangle \longrightarrow^* \langle E, \sigma'\rangle \ \wedge\ (S_1 \Box S_2, \sigma)\downarrow\}\}$

=

$$\{\sigma|P.\{\sigma'|(\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle \ \vee \ \langle S_2,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle) \ \wedge \ ((S_1,\sigma)\downarrow \ \wedge (S_2,\sigma)\downarrow)\}\}$$

$=$ {definition of Op }

$$\{\sigma|(P.Op.S_1.\sigma \ \wedge \ (S_2,\sigma)\downarrow) \ \wedge \ (P.Op.S_2.\sigma \ \wedge \ (S_1,\sigma)\downarrow)\}$$

$=$

$$\{\sigma|P.Op.S_1.\sigma \ \wedge \ P.Op.S_2.\sigma\}$$

$=$ { induction }

$$wp.S_1.P \wedge wp.S_2.P$$

$=$

$$wp.S_1 \Box S_2.$$

- $S_1 \Diamond S_2$) $\{\sigma|P.Op.S_1 \Diamond S_2.\sigma\}$

$=$ { definition of Op, $P.\ \perp =$ **false** }

$$\{\sigma|P.\{\sigma'|\langle S_1 \Diamond S_2,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle \ \wedge \ (S_1 \Diamond S_2,\sigma)\downarrow\}\}$$

$=$ { definition of $\longrightarrow$ }

$$\{\sigma|P.\{\sigma'|(\langle S_1,\sigma\rangle \longrightarrow \langle E,\sigma'\rangle \ \vee \ \langle S_1 \triangle (S_2,\sigma),\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle) \ \wedge \ (S_1 \Diamond S_2,\sigma)\downarrow\}\}$$

$=$ { definition of $\longrightarrow$ }

$$\{\sigma|P.\{\sigma'|(\langle S_1,\sigma\rangle \longrightarrow \langle E,\sigma'\rangle \ \vee \ \langle S_1,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle \ \wedge \ \neg\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\delta\rangle) \vee$$

$$\vee \ (\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\delta\rangle \ \wedge \ \langle S_2,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle) \ \wedge \ (S_1 \Diamond S_2,\sigma)\downarrow\}\}$$

$=$

$$\{\sigma|P.\{\sigma'|(\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle \wedge \ \neg\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\delta\rangle) \vee$$

$$\vee \ (\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\delta\rangle \ \wedge \ \langle S_2,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle) \ \wedge \ (S_1 \Diamond S_2,\sigma)\downarrow\}\}$$

$=$

$$\{\sigma|P.\{\sigma'|(\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle \ \wedge \ \neg\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\delta\rangle \ \wedge \ (S_1,\sigma)\downarrow) \vee$$

$$\vee \ (\langle S_1,\sigma\rangle \longrightarrow^* \langle E,\delta\rangle \ \wedge \ \langle S_2,\sigma\rangle \longrightarrow^* \langle E,\sigma'\rangle \ \wedge \ (S_1,\sigma)\downarrow \ \wedge (S_2,\sigma)\downarrow)\}\}$$

$=$ { definition of Op }

$$\{\sigma|P.\{\sigma'|(\sigma' \in Op.S_1.\sigma \ \wedge \ Op.S_1.\sigma \neq \emptyset) \vee$$

$$\vee\ (Op.S_1.\sigma = \emptyset\ \wedge\ \sigma' \in Op.S_2.\sigma)\}\}$$

$=$

$$\{\sigma | P.Op.S_1.\sigma\ \wedge\ Op.S_1.\sigma \neq \emptyset\}\ \cup\ \{\sigma | Op.S_1.\sigma = \emptyset\ \wedge\ P.Op.S_2.\sigma\}$$

$=$ { using induction on corollary 2.6 }

$$(wp.S_1.P \wedge \neg wp.S_1.\mathbf{false}) \vee (wp.S_1.\mathbf{false} \wedge wp.S_2.P)$$

$=$ { because $wp.S_1.\mathbf{false} \Rightarrow wp.S_1.P$ }

$$(wp.S_1.P \wedge \neg wp.S_1.\mathbf{false}) \vee (wp.S_1.P \wedge wp.S_1.\mathbf{false} \wedge wp.S_2.P)$$

$=$

$$wp.S_1.P \wedge (\neg wp.S_1.\mathbf{false} \vee (wp.S_1.\mathbf{false} \wedge wp.S_2.P))$$

$=$

$$wp.S_1.P \wedge (wp.S_1.\mathbf{false} \Rightarrow wp.S_2.P)$$

$=$

$$wp.S_1 \diamond S_2.P.$$

□