Metric Semantics for the Input/Output Behaviour of Sequential Programs

Joost N. Kok
University of Utrecht*

Abstract

As in the book *Mathematical Theory of Program Correctness* by J.W. de Bakker we assign both operational and denotational semantics to a language with while and one with simultaneous recursion. The main difference is that we use tools from metric topology instead of order theory.

1 Introduction

The book *Mathematical Theory of Program Correctness* by J.W. de Bakker ([dB80]) has become a standard work in the field of the semantics and correctness of sequential programming. The mathematical basis of this book is order theory. An important role is played by the following fixed point theorem: continuous functions on a complete partial ordering have least fixed points. This theorem is applied in fixed point definitions: for example in the definition of a semantic model for a language with recursive procedures. This is a typical situation in which functions have more than one fixed point.

In later work of J.W. de Bakker metric topology is used as the main mathematical tool for semantic theories, for example in [dBZ82] or [dBM88]. The metric equivalent of the fixed point theorem for continuous functions is Banach's theorem: a contraction on a complete metric space has a unique fixed point. This theorem is works very well in situations where functions have have unique fixed points.

Both fixed point theorems state that the (least) fixed point can be obtained by an iteration procedure. in the first case as the least upperbound of a chain starting in the least element of the complete partial ordering, in the second case as the limit of a Cauchy sequence starting from an arbitrary element of the metric space. In both cases each next element is obtained by applying the funtion to the previous element.

It has been an open question whether we can apply metric theory in non unique fixed point situations. In this paper we take as a starting point chapters three and four of *Mathematical Theory of Program Correctness* and show how we can treat the semantic theory also in a metric way. The first chapter deals with a language containing

*Department of Computer Science, University of Utrecht, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.

assignment, sequential composition, if statement and while statement. The second chapter gives semantics to a language in which in addition we have simultaneous recursion. Our goal is to give to both languages input/output semantics (a function from the initial state to (if it exists) the final state): an operational one by using transition systems and denotational with the aid of metric topology. In both cases we derive two independent characterizations of the denotational semantics:

1. it is the limit of a Cauchy sequence,
2. it is the fixed point with the least distance to the nowhere defined function.

We also study the equivelence between the different semantic models. It turns out that the proofs of the equivalences are a mixture of the proofs in *Mathematical Theory of Program Correctness* and [KR88].

2 While statements

We start by giving the set of statements. The notation $(x \in)X$ introduces the set X with typical element x ranging over X. The set of integers and the set of booleans are denoted by $\mathbb{N}$ and $\mathbb{B}$ respectively.

Definition 2.1 *Let* $(b \in)$ *BExp,* $(x, y \in)$ *Var, and* $(s \in)$ *Exp be sets. Let* $(S \in)$ *Stat be the set statements specified by*

$$S ::= x := s \mid S_1; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \mid \textbf{while } b \textbf{ do } S \textbf{ od}.$$

Next we introduce the notion of a state:

Definition 2.2 *The set* Σ *of states is given by* $\Sigma = Var \to \mathbb{N}$.

We use the notation $\sigma[x := \alpha]$, with $\alpha \in \mathbb{N}$, for a variant of σ, i.e., for the state which is defined by

$$\sigma[x := \alpha] = \lambda y. \begin{cases} \alpha & \text{if } x = y \\ \sigma(y) & \text{otherwise} \end{cases}$$

We assume given functions $\mathcal{V} : Exp \to \Sigma \to \mathbb{N}$ and $\mathcal{W} : BExp \to \Sigma \to \mathbb{B}$. We now give the definition of the operational semantics. It is based on a transition system. Here, a transition is a fourtuple in $Stat \times \Sigma \times (Stat \cup \{E\}) \times \Sigma$, written in the notation

$$(S, \sigma) \to (S', \sigma')$$

or

$$(S, \sigma) \to (E, \sigma').$$

The symbol E is called the empty statement and stands for termination. We present a formal transition system T which consists of axioms (in one of the two forms above) or rules, in the form

$$\frac{(S_1,\sigma_1) \to (S_1',\sigma_1')}{(S_2,\sigma_2) \to (S_2',\sigma_2')}$$

in which both S_1' and S_2' can be replaced by E. Transitions which are given as axioms hold by definition. Moreover, a transition which is the consequence of a rule holds whenever it can be established that its premise holds.

Axioms:

$$(x := s,\sigma) \to (E,\sigma[x := \mathcal{V}(s)(\sigma)])$$

$$(\textbf{while } b \textbf{ do } S \textbf{ od},\sigma) \to (E,\sigma) \textit{ if } \mathcal{W}(b)(\sigma) = \textit{ff}$$

Rules:

$$\frac{(S_1,\sigma) \to (S_1',\sigma')}{(S_1;S,\sigma) \to (S_1';S,\sigma')}$$
$$(\textbf{while } b \textbf{ do } S_1 \textbf{ od},\sigma) \to (S_1';\textbf{while } b \textbf{ do } S_1 \textbf{ od},\sigma') \textit{ if } \mathcal{W}(b)(\sigma) = \textit{tt}$$
$$(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S \textbf{ fi},\sigma) \to (S_1',\sigma') \textit{ if } \mathcal{W}(b)(\sigma) = \textit{tt}$$
$$(\textbf{if } b \textbf{ then } S \textbf{ else } S_1 \textbf{ fi},\sigma) \to (S_1',\sigma') \textit{ if } \mathcal{W}(b)(\sigma) = \textit{ff}$$

$$\frac{(S_1,\sigma) \to (E,\sigma')}{(S_1;S,\sigma) \to (S,\sigma')}$$
$$(\textbf{while } b \textbf{ do } S_1 \textbf{ od},\sigma) \to (\textbf{while } b \textbf{ do } S_1 \textbf{ od},\sigma') \textit{ if } \mathcal{W}(b)(\sigma) = \textit{tt}$$
$$(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S \textbf{ fi},\sigma) \to (E,\sigma') \textit{ if } \mathcal{W}(b)(\sigma) = \textit{tt}$$
$$(\textbf{if } b \textbf{ then } S \textbf{ else } S_1 \textbf{ fi},\sigma) \to (E,\sigma') \textit{ if } \mathcal{W}(b)(\sigma) = \textit{ff}$$

Let $\Sigma_\perp = \Sigma \cup \{\perp\}$, where $\perp$ is a special. From now on, σ ranges over $\Sigma_\perp$. We proceed with the definition of the operational semantics $\mathcal{O}$.

Definition 2.3 *The mapping $\mathcal{O} : Stat \to \Sigma_\perp \to \Sigma_\perp$ is given by*

$$\mathcal{O}(S)(\sigma) = \begin{cases} \perp & \sigma = \perp \\ \sigma' & (S,\sigma) \to^* (E,\sigma') \\ \perp & \textit{otherwise} \end{cases}$$

The next step is the development of a metric denotational model. We have to assume that the set $\Sigma_\perp$ is countable, i.e. $\Sigma_\perp = \{\sigma_1,\sigma_2,\ldots\}$. A metric d on $\Sigma_\perp \to \Sigma_\perp$ is defined as follows: Let $f,g \in \Sigma_\perp \to \Sigma_\perp$.

$$d(f,g) = \sum_{i=1}^{\infty} \begin{cases} 0 & f(\sigma_i) = g(\sigma_i) \\ 10^{-i} & f(\sigma_i) \neq g(\sigma_i). \end{cases}$$

The denotational semantics $\mathcal{D} : Stat \to \Sigma_\perp \to \Sigma_\perp$ is given in

Definition 2.4

1. $\mathcal{D}(x := s) = \lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ \sigma[x := \mathcal{V}(s)(\sigma)] & \textit{otherwise} \end{cases}$

2. $\mathcal{D}(S_1; S_2) = \mathcal{D}(S_2) \circ \mathcal{D}(S_1)$

3. $\mathcal{D}(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}) = \lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ \mathcal{D}(S_1)(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \mathcal{D}(S_2)(\sigma) & \mathcal{W}(b)(\sigma) = ff \end{cases}$

4. $\mathcal{D}(\textbf{while } b \textbf{ do } S \textbf{ od}) = \lim_{i\to\infty} \phi_i$ *where*

$$\phi_0 = \lambda\sigma.\bot$$

$$\phi_{i+1} = \lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ (\phi_i \circ \mathcal{D}(S))(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \sigma & \mathcal{W}(b)(\sigma) = ff \end{cases}$$

Lemma 2.5 *The sequence $(\phi_i)_i$ is a Cauchy sequence in $\Sigma_\bot \to \Sigma_\bot$.*

Proof: Follows directly from $\forall i[\phi_i(\sigma) = \sigma' \wedge \sigma' \neq \bot \Rightarrow \phi_{i+1}(\sigma) = \sigma']$. □

Theorem 2.6 *Let $\Psi : (\Sigma_\bot \to \Sigma_\bot) \to (\Sigma_\bot \to \Sigma_\bot)$ be defined as follows.*

$$\Psi(F)(\sigma) = \lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ (F \circ \mathcal{D}(S))(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \sigma & \mathcal{W}(b)(\sigma) = ff. \end{cases}$$

Let $\phi = \mathcal{D}(\textbf{while } b \textbf{ do } S \textbf{ od})$. Then ϕ is the fixed point of Ψ with the smallest distance to $\lambda\sigma.\bot$.

We continue with the derivation of the equivalence $\mathcal{O} = \mathcal{D}$. The mapping $\Phi : (Stat \to (\Sigma_\bot \to \Sigma_\bot)) \to (Stat \to (\Sigma_\bot \to \Sigma_\bot))$ is given in

Definition 2.7
$$\Phi(F)(S)(\sigma) = \begin{cases} \bot & \sigma = \bot \\ \sigma' & (S, \sigma) \to (E, \sigma') \\ F(S')(\sigma') & (S, \sigma) \to (S', \sigma') \end{cases}$$

An important step in the proof that $\mathcal{O} = \mathcal{D}$ holds on *Stat* is the following lemma:

Lemma 2.8

1. $\Phi(\mathcal{D})(S_1; S_2) = \mathcal{D}(S_2) \circ \Phi(\mathcal{D})(S_1)$

2. $\Phi(\mathcal{D})(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}) = \lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ \Phi(\mathcal{D})(S_1)(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \Phi(\mathcal{D})(S_2)(\sigma) & \mathcal{W}(b)(\sigma) = ff \end{cases}$

3. $\Phi(\mathcal{D})(\textbf{while } b \textbf{ do } S \textbf{ od}) = \lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ (\mathcal{D}(\textbf{while } b \textbf{ do } S \textbf{ od}) \circ \Phi(\mathcal{D})(S))(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \sigma & \mathcal{W}(b)(\sigma) = ff. \end{cases}$

We have the following

Corollary 2.9 *$\mathcal{D}$ is a fixed point of Φ.*

The equivalence $\mathcal{O} = \mathcal{D}$ follows by the following

Theorem 2.10 *For all $S \in Stat$, $\sigma, \sigma' \in \Sigma$*

$$(S,\sigma) \to^* (E,\sigma') \Leftrightarrow \mathcal{D}(S)(\sigma) = \sigma' \wedge \sigma' \neq \bot$$

Proof:

$\Rightarrow$:

Assume $(S,\sigma) \to^n (E,\sigma')$. Using the corollary we have

$$\mathcal{D} = \Phi(\mathcal{D}) = \cdots = \Phi^n(\mathcal{D}) =$$

$$\lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ \sigma' & (S,\sigma) \to^k (E,\sigma') \wedge k \leq n \\ \mathcal{D}(S')(\sigma') & (S,\sigma) \to^n (E,\sigma'). \end{cases}$$

Hence $\mathcal{D}(S)(\sigma) = \sigma'$.

$\Leftarrow$:

We prove the result by induction on the complexity of the statement S. We distinguish the following cases:

1. $S \equiv x := s$:

   $$\mathcal{D}(S)(\sigma) = \sigma' \wedge \sigma' \neq \bot \Rightarrow \sigma' = \sigma[x := \mathcal{V}(s)(\sigma)]$$

   and

   $$(x := s, \sigma) \to (E, \sigma[x := \mathcal{V}(s)(\sigma)])$$

2. $S \equiv S_1; S_2$:

   $$\mathcal{D}(S_1; S_2)(\sigma) = \sigma' \wedge \sigma' \neq \bot \Rightarrow$$

   $$\exists\sigma''[\sigma'' = \mathcal{D}(S_1)(\sigma) \wedge \sigma' = \mathcal{D}(S_2)(\sigma'')] \Rightarrow$$

   $$\exists\sigma''[(S_1,\sigma) \to^* (E,\sigma'') \wedge (S_2,\sigma'') \to^* (E,\sigma')] \Rightarrow$$

   $$(S_1; S_2, \sigma) \to^* (E, \sigma')$$

3. $S \equiv$ **while** b **do** S_1 **od**:

   $$\mathcal{D}(\textbf{while } b \textbf{ do } S_1 \textbf{ od})(\sigma) = \sigma' \wedge \sigma' \neq \bot \Rightarrow$$

$$(\lim_{i\to\infty}\phi_i)(\sigma) = \sigma' \wedge \sigma' \neq \perp \Rightarrow$$

$$\exists k[\phi_k)(\sigma) = \sigma' \wedge \sigma' \neq \perp] \Rightarrow$$

$$\exists k[\sigma' = \underbrace{\mathcal{D}(S_1) \circ \cdots \mathcal{D}(S_1)}_{k}(\sigma) \wedge \mathcal{W}(b)(\sigma') = \mathit{ff} \wedge$$

$$\forall j : j < k[\mathcal{W}(b)\underbrace{\mathcal{D}(S_1) \circ \cdots \mathcal{D}(S_1)}_{j}(\sigma) = \mathit{tt}] \wedge \sigma' \neq \perp] \Rightarrow$$

$$\exists \sigma_1, \ldots, \sigma_{k+1}[$$

$$\sigma_1 = \sigma \wedge \sigma_2 = \mathcal{D}(S_1)(\sigma_1) \cdots \wedge \sigma_{k+1} = \mathcal{D}(S_1)(\sigma_k) \wedge \sigma_{k+1} = \sigma' \wedge$$

$$\mathcal{W}(b)(\sigma') = \mathit{ff} \wedge \forall i \in \{1, \ldots, k\}[\mathcal{W}(b)(\sigma_i) = \mathit{tt}]] \Rightarrow$$

$$\exists \sigma_1, \ldots, \sigma_{k+1}[$$

$$(S_1, \sigma_1) \rightarrow^* (E, \sigma_2) \wedge \mathcal{W}(b)(\sigma_1) = \mathit{tt} \wedge$$

$$(S_1, \sigma_2) \rightarrow^* (E, \sigma_3) \wedge \mathcal{W}(b)(\sigma_2) = \mathit{tt} \wedge$$

$$\ldots$$

$$(S_1, \sigma_k) \rightarrow^* (E, \sigma_{k+1}) \wedge \mathcal{W}(b)(\sigma_k) = \mathit{tt} \wedge$$

$$\mathcal{W}(b)(\sigma_{k+1}) = \mathit{ff}] \Rightarrow$$

$$(\textbf{while } b \textbf{ do } S_1 \textbf{ od}, \sigma) \rightarrow^* (E, \sigma').$$

□

3 Recursion

We now present the syntax for a language with recursion. Again, we use S to range over the set of statements *Stat*. The set of statements is extended with a new syntactic category, the set $(P \in) PVar$ of procedure variables.

Definition 3.1

1. *Let* $(S \in)$ *Stat be the set statements specified by*

$$S \ ::= \ x := s \mid S_1; S_2 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \mid P.$$

2. *The set* $(G \in)$ *GStat of guarded statements is given by*

$$G \ ::= \ x := s \mid G_1; S_2 \mid \textbf{if } b \textbf{ then } G_1 \textbf{ else } G_2 \textbf{ fi}$$

3. *The set* $(D \in)$ *Decl of declarations consists of n-tuples* $D \equiv P_1 \Leftarrow G_1, \ldots, P_n \Leftarrow G_n$.

4. *The set* $(R \in)$ *Prog of programs consists of tuples* $R \equiv D \mid S$.

We use simultaneous recursion rather than using so-called μ-constructs. In procedure declarations we only use guarded statements. The set of guarded statements is a subset of the set of statements. Declarations with unguarded statements can be made guarded with *skip*-statements. For the input/output behaviour this does not make any difference.

Now, a transition is a fourtuple in $Prog \times \Sigma \times (Prog \cup \{E\}) \times \Sigma$, written in the notation

$$(R, \sigma) \rightarrow (R', \sigma')$$

or

$$(R, \sigma) \rightarrow (E, \sigma').$$

The transition system which generates the transition relation is given in

Axioms:

$$(x := s, \sigma) \rightarrow (E, \sigma[x := \mathcal{V}(s)(\sigma)])$$

Rules:

$$\frac{(D \mid S_1, \sigma) \rightarrow (D \mid S_1', \sigma')}{(D \mid S_1; S, \sigma) \rightarrow (D \mid S_1'; S, \sigma')}$$

$$(D \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S \textbf{ fi}, \sigma) \rightarrow (D \mid S_1', \sigma') \textit{ if } \mathcal{W}(b)(\sigma) = tt$$

$$(D \mid \textbf{if } b \textbf{ then } S \textbf{ else } S_1 \textbf{ fi}, \sigma) \rightarrow (D \mid S_1', \sigma') \textit{ if } \mathcal{W}(b)(\sigma) = ff$$

$$(\ldots, P \Leftarrow S_1, \ldots \mid P, \sigma) \rightarrow (\ldots, P \Leftarrow S_1, \ldots \mid S_1', \sigma')$$

$$\frac{(D \mid S_1, \sigma) \rightarrow (E, \sigma')}{(D \mid S_1; S, \sigma) \rightarrow (D \mid S, \sigma')}$$

$$(D \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S \textbf{ fi}, \sigma) \rightarrow (E, \sigma') \textit{ if } \mathcal{W}(b)(\sigma) = tt$$

$$(D \mid \textbf{if } b \textbf{ then } S \textbf{ else } S_1 \textbf{ fi}, \sigma) \rightarrow (E, \sigma') \textit{ if } \mathcal{W}(b)(\sigma) = ff$$

$$(\ldots, P \Leftarrow S_1, \ldots \mid P, \sigma) \rightarrow (E, \sigma')$$

We next define the operational semantics.

Definition 3.2 *The mapping* $\mathcal{O} : Prog \rightarrow \Sigma_\perp \rightarrow \Sigma_\perp$ *is given by*

$$\mathcal{O}(R)(\sigma) = \begin{cases} \perp & \sigma = \perp \\ \sigma' & (R, \sigma) \rightarrow^* (E, \sigma') \\ \perp & \textit{otherwise} \end{cases}$$

We introduce the notion of environment which is used to store and retrieve meanings of procedure variables. Let $\Gamma = PVar \to \Sigma_\perp \to \Sigma_\perp$ be the set of environments, and let $\gamma \in \Gamma$. We write $\gamma[P_i := \phi_i]$ for a variant of γ which is like γ but with $\gamma(P_i) = \phi_i$. We are now in a position to define the denotational semantics for $R \in Prog$. The denotational semantics $\mathcal{D} : \Gamma \to Stat \to \Sigma_\perp \to \Sigma_\perp$ is given in

Definition 3.3

1. $\mathcal{D}(\gamma)(x := s) = \lambda\sigma.\begin{cases} \perp & \sigma = \perp \\ \sigma[x := \mathcal{V}(s)(\sigma)] & \textit{otherwise} \end{cases}$
2. $\mathcal{D}(\gamma)(S_1; S_2) = \mathcal{D}(\gamma)(S_2) \circ \mathcal{D}(\gamma)(S_1)$
3. $\mathcal{D}(\gamma)(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}) = \lambda\sigma.\begin{cases} \perp & \sigma = \perp \\ \mathcal{D}(\gamma)(S_1)(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \mathcal{D}(\gamma)(S_2)(\sigma) & \mathcal{W}(b)(\sigma) = ff \end{cases}$
4. $\mathcal{D}(\gamma)(P) = \gamma(P)$

We define a metric d on Γ by putting

$$d(\gamma_1, \gamma_2) = \sup\{d(\gamma_1(P), \gamma_2(P)) : P \in PVar\}.$$

This metric turns Γ into a complete metric space.

Choose an arbitrary declaration $D \equiv P_1 \Leftarrow G_1, \ldots, P_n \Leftarrow G_n$. Let the mapping $\Psi : \Gamma \to \Gamma$ be given by

$$\Psi(\gamma) = \gamma[P_i := \mathcal{D}(\gamma)(G_i)].$$

We have the following lemma.

Lemma 3.4 *The sequence $(\Psi^j(\lambda P.\lambda\sigma.\perp))_j$ is a Cauchy sequence.*

Proof Follows from

$$\forall j[\Psi^j(\lambda P.\lambda\sigma.\perp)(P)(\sigma) = \sigma' \wedge \sigma' \neq \perp \Rightarrow \Psi^{j+1}(\lambda P.\lambda\sigma.\perp)(P)(\sigma) = \sigma'].$$

□

Theorem 3.5 *Let $\gamma_D = \lim_{i\to\infty} \Psi^i(\lambda P.\lambda\sigma.\perp)$. Then γ_D is the fixed point of Ψ with the least distance to $\lambda P.\lambda\sigma.\perp$.*

The mapping $\mathcal{M} : Prog \to \Sigma_\perp \to \Sigma_\perp$ is given as follows:

$$\mathcal{M}(D \mid S) = \mathcal{D}(\gamma_D)(S).$$

We turn to the equivalence $\mathcal{O} = \mathcal{M}$. The mapping $\Phi : (Prog \to (\Sigma_\perp \to \Sigma_\perp)) \to (Prog \to (\Sigma_\perp \to \Sigma_\perp))$ is given in

Definition 3.6 $$\Phi(F)(R)(\sigma) = \begin{cases} \bot & \sigma = \bot \\ \sigma' & (R,\sigma) \to (E,\sigma') \\ F(R')(\sigma') & (R,\sigma) \to (R',\sigma') \end{cases}$$

An important step in the proof that $\mathcal{O} = \mathcal{M}$ holds on *Prog* is the following lemma:

Lemma 3.7

1. $\Phi(\mathcal{M})(D \mid S_1; S_2) = \mathcal{M}(D \mid S_2) \circ \Phi(\mathcal{M})(D \mid S_1)$
2. $\Phi(\mathcal{M})(\ldots, P \Leftarrow G, \ldots \mid P) = \Phi(\mathcal{M})(\ldots, P \Leftarrow G, \ldots \mid G)$
3. $\Phi(\mathcal{M})(D \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}) = \lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ \Phi(\mathcal{M})(D \mid S_1)(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \Phi(\mathcal{M})(D \mid S_2)(\sigma) & \mathcal{W}(b)(\sigma) = ff \end{cases}$

We have the following

Corollary 3.8 *$\mathcal{M}$ is a fixed point of Ψ.*

Proof: We show for any $D \in Decl$ and $S \in Stat$

$$(*)\ \Phi(\mathcal{M})(D \mid S) = \mathcal{M}(D \mid S)$$

The proof proceeds in two stages; first we show $(*)$ for $G \in GStat (\subseteq Stat)$ and next for $S \in Stat$. We prove the result for $G \in GStat (S \in Stat)$ by induction on the complexity of the statement $G(S)$.

1. We only treat the cases $G \equiv G_1; S$ and $G \equiv \textbf{if } b \textbf{ then } G_1 \textbf{ else } G_2 \textbf{ fi}$.

   $G \equiv G_1; S$:

   $$\Phi(\mathcal{M})(D \mid G_1; S) = [lemma]$$

   $$\mathcal{M}(D \mid S) \circ \Phi(\mathcal{M})(D \mid G_1) = [induction]$$

   $$\mathcal{M}(D \mid S) \circ \mathcal{M}(D \mid G_1) =$$

   $$\mathcal{M}(D \mid G_1; S).$$

   $G \equiv \textbf{if } b \textbf{ then } G_1 \textbf{ else } G_2 \textbf{ fi}$:

   $$\Phi(\mathcal{M})(D \mid \textbf{if } b \textbf{ then } G_1 \textbf{ else } G_2 \textbf{ fi}) = [lemma]$$

   $$\lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ \Phi(\mathcal{M})(D \mid G_1)(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \Phi(\mathcal{M})(D \mid G_2)(\sigma) & \mathcal{W}(b)(\sigma) = ff \end{cases} = [induction]$$

$$\lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ \mathcal{M}(D \mid G_1)(\sigma) & \mathcal{W}(b)(\sigma) = tt \\ \mathcal{M}(D \mid G_2)(\sigma) & \mathcal{W}(b)(\sigma) = ff \end{cases} =$$

$$\mathcal{M}(D \mid \textbf{if } b \textbf{ then } G_1 \textbf{ else } G_2 \textbf{ fi}).$$

2. We only treat the case

   $S \equiv P$:

   $$\Phi(\mathcal{M})(\ldots, P \Leftarrow G, \ldots \mid P) = [\textit{lemma}]$$

   $$\Phi(\mathcal{M})(\ldots, P \Leftarrow G, \ldots \mid G) = [G \in \textit{GStat}]$$

   $$\mathcal{M}(\ldots, P \Leftarrow G, \ldots \mid G) = [\textit{induction}]$$

   $$\mathcal{M}(\ldots, P \Leftarrow G, \ldots \mid P).$$

□

Theorem 3.9 $(R, \sigma) \rightarrow^* (E, \sigma') \Leftrightarrow \mathcal{M}(R)(\sigma) = \sigma' \wedge \sigma' \neq \bot$.

Proof:

$\Rightarrow$:

Assume $(R, \sigma) \rightarrow^n (E, \sigma')$. Using the corollary we have

$$\mathcal{M} = \Phi(\mathcal{M}) = \cdots = \Phi^n(\mathcal{M}) =$$

$$\lambda\sigma.\begin{cases} \bot & \sigma = \bot \\ \sigma' & (R, \sigma) \rightarrow^k (E, \sigma') \wedge k \leq n \\ \mathcal{D}(R')(\sigma') & (R, \sigma) \rightarrow^n (E, \sigma'). \end{cases}$$

Hence $\mathcal{M}(S)(\sigma) = \sigma'$.

$\Leftarrow$: Let, for eack k, $\gamma_k = \Psi^k(\lambda P.\lambda\sigma.\bot)$. Note that $\lim_{k\to\infty} \gamma_k = \gamma_D$. We prove for all k, S, σ, σ'

$$\sigma' = \mathcal{D}(\gamma_k)(S)(\sigma) \wedge \sigma' \neq \bot \Rightarrow (D \mid S, \sigma) \rightarrow^* (E, \sigma').$$

We prove it by induction on the tuples made up of k and the complexity of the statement S. As the ordering on such tuples we take the lexicographical ordering. We only treat the cases $S \equiv S_1; S_2$ and $S \equiv P$.

$S \equiv S_1; S_2$:

$$\mathcal{D}(\gamma_k)(S_1; S_2) =$$

$$\mathcal{D}(\gamma_k)(S_2) \circ \mathcal{D}(\gamma_k)(S_1).$$

Because

$$\sigma' = \mathcal{D}(\gamma_k)(S_1; S_2)(\sigma) \wedge \sigma' \neq \bot$$

we can find a σ'' such that

$$\sigma'' = \mathcal{D}(\gamma_k)(S_1)(\sigma) \wedge \sigma'' \neq \bot$$

$$\sigma' = \mathcal{D}(\gamma_k)(S_2)(\sigma'') \wedge \sigma' \neq \bot.$$

By induction

$$(D \mid S_1, \sigma) \rightarrow^* (E, \sigma'')$$

$$(D \mid S_2, \sigma'') \rightarrow^* (E, \sigma')$$

and hence

$$(D \mid S_1; S_2, \sigma'') \rightarrow^* (E, \sigma').$$

$S \equiv P$: Let G the body of P.

$$\mathcal{D}(\gamma_k)(P) =$$

$$\gamma_k(P) =$$

$$\mathcal{D}(\gamma_{k_1})(G).$$

Hence by induction

$$(D \mid G, \sigma) \rightarrow^* (E, \sigma')$$

and this implies

$$(D \mid P, \sigma) \rightarrow^* (E, \sigma').$$

□

4 Conclusion

We hope that this paper shows that we also can handle the input/output semantics of sequential languages with the help of metric topology. In this case, the difference between using order theory or metric topology is not very big. The proofs are in both cases very similar.

References

[dB80] J.W. de Bakker. *Mathematical Theory of Program Correctness*. Prentice/Hall, 1980.

[dBM88] J.W. de Bakker and J.-J.Ch. Meyer. Metric semantics for concurrency. *BIT*, 28:504–529, 1988.

[dBZ82] J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Inform. and Control*, 54:70–120, 1982.

[KR88] J.N. Kok and J.J.M.M. Rutten. Contractions in comparing concurrency semantics. In T. Lepistö and A. Salomaa, editors, *Proc. 15th International Colloquium Automata, Languages and Programming*, pages 317–332, Springer Verlag, 1988. Lecture Notes in Computer Science 317.