Reihe Informatik

4/89

Some Comments on CPO-Semantics and Metric Space Semantics for Imperative Languages

by

Mila E. Majster-Cederbaum

September 1989

In the past various approaches for the specification of semantics of programming languages have been proposed [1, 5, 7, 12, 15, 23–26, 28–30, 33–35]. Recently, in the context of parallelism and nondeterminism the approach of [11] has attracted growing interest. Further investigations are found e.g. in [2, 3, 4, 9, 10, 19–21].

In particular for Hoare's language of communicating sequential processes (CSP) [16] various semantic specifications have been proposed, two of which are sketched below. Hoare's approach [12] is based on the theory of complete partial orders (cpo) and is an extension of Scott's denotational semantics for sequential programming languages [33, 34]. In this approach the basic idea was to associate with a (deterministic sequential) program X a function f_X called the "denotation of X". In the case of very simple programs, involving assignment, if– and while–constructions the meaning associated with a program X is a function $f_X : \Sigma \to \Sigma$, where Σ is a suitable set of states. The necessity to model the semantics of iteration, recursion and self–application triggered the need for a framework for the solution of domain equations and for the solution of fixed point equations within a particular domain. The result was a formalism based on complete partial orders. In the case of a nondeterministic sequential program Y the approach was extended in [27, 36] to associate a meaning f_Y that is a mapping from Σ to some power set construction of Σ. Finally, in the case of CSP the meaning of a parallel command $Z = [Y_1 || \ldots || Y_n]$, where Y_i is a sequential nondeterministic process with states in Σ_i, is given by first producing a "history tree" for each process Y_i started in state σ_i that records local and global nondeterminism but does not record intermediate states. In a second step these trees are evaluated by a "binding function" and as a result a function f_Z from $\Sigma_1 \times \ldots \times \Sigma_n$ to a power domain construction of $\Sigma_1 \times \ldots \times \Sigma_n$ is obtained [12].

In the approach of de Bakker and Zucker [11] the role of *cpo*'s is taken by complete metric spaces. The approach is – as the authors state – denotational in style but operational in spirit. For a process all possible action sequences are recorded. Let us briefly sketch the approach. Let an example language be given by the following syntactical description

$$S \to \text{skip} \mid x := e \mid (S_1; S_2) \mid b \mid \text{while } b \text{ do } S \quad .$$

We define a process domain by considering the equation

$$\text{(I)} \qquad P = \{p_0\} \cup \left(\Sigma \to \wp_c(\Sigma \times P)\right) \quad .$$

Here P stands for a complete metric space, Σ is a set and $\wp_c(\cdot)$ denotes the set of all closed subsets. Applying the theory developed in [4, 11, 19, 20, 21] one can conclude that there is a unique complete metric space P that satisfies the above equation. The meaning of a program X in the sample language will be defined as an element in P, i.e. either p_0 or a function $f_X : \Sigma \to \wp_c(\Sigma \times P)$. In particular we put: Let Me be the meaning function associating a meaning with every program X. Then

$$\text{Me}(\text{skip}) = \lambda\sigma.\{\langle\sigma, p_0\rangle\}$$

$$\mathbf{Me}(x := E) = \lambda\sigma.\{\langle \sigma_x^{\mathcal{V}(E)(\sigma)}, p_0\rangle\}$$

$$\mathbf{Me}(b) = \lambda\sigma.\text{if } \mathcal{W}(b)(\sigma) \text{ then } \{\langle\sigma, p_0\rangle\} \text{ else } \emptyset$$

$$\mathbf{Me}\big((S_1; S_2)\big) = \mathbf{Me}(S_2) \circ \mathbf{Me}(S_1) \qquad \text{etc.}$$

in analogy to [11].

In this definition V is a set of values, $\Sigma = \mathrm{Var} \to V$ the set of states and

$$\mathcal{V} : \mathrm{Exp} \to (\Sigma \to V)$$

$$\mathcal{W} : \mathrm{Test} \to (\Sigma \to W)$$

describe the meaning of expressions, respectively tests. Moreover, for $v \in V$

$$\sigma_x^v : \mathrm{Var} \to V$$

$$\sigma_x^v(y) = \begin{cases} \sigma(y) & y \neq x \\ v & y = x \end{cases}$$

For $p, q \in \{p_0\} \cup \big(\Sigma \to \wp_c(\Sigma \times P)\big)$, $X \in \wp_c(\Sigma \times P)$ the operation $\circ$ is defined in analogy to [11] by

$$p \circ p_0 = p$$

$$p \circ \langle\sigma, q\rangle = \langle\sigma, p \circ q\rangle$$

$$p \circ X = \{p \circ x : x \in X\}$$

$$p \circ \lambda\sigma.X = \lambda\sigma.(p \circ X)$$

If we consider e.g. the program

$$S \equiv x := E_1;\ y := E_2$$

then

$$\mathbf{Me}[S] = \lambda\sigma\left\{\left\langle {\sigma_x}^{\mathcal{V}(E_1)(\sigma)}, \lambda\sigma'\left\{\langle {\sigma'_y}^{\mathcal{V}(E_2)(\sigma')}, p_0\rangle\right\}\right\rangle\right\}$$

so that the sequence of states that has been passed by the execution of S starting from state σ is obtained by

$$\sigma_x^{\mathcal{V}(E_1)(\sigma)}, \left(\sigma_x^{\mathcal{V}(E_1)(\sigma)}\right)_y^{\mathcal{V}(E_2)(\sigma_x^{\mathcal{V}(E_1)(\sigma)})}.$$

Basicly, this kind of semantics keeps track of all states that have been reached by executing a (sequential deterministic) program.

In contrast to this, the denotational approach of Scott and Strachey was designed to deal with the problem how to associate with a (deterministic sequential) program a meaning that is basicly a function from the set of states to itself ignoring any "intermediate" states.

Problem 1

The first question we are interested in is, if the metric space approach is inherently operational, in the sense that it traces all the steps that are performed during the execution, or if we could

define the meaning of a program X – based on the metric space approach – that disregards the intermediate states. A first observation in this context is that – in the *cpo* approach – we have the $\perp$ (bottom) element for the treatment of partial functions, whereas – in the case of metric spaces – it is not evident how to handle partiality of functions. In the operational definition using metric spaces as given by [11] nonterminating computations, that would cause a meaning function from Σ to Σ to be partial, are handled by limits of appropriate sequences.

One option here is obviously to enrich the set of states Σ by an element ω. If (Σ, d) is a complete metric space so evidently $\Sigma' = \Sigma \cup \{\omega\}$ can be made one as well. Another option might be to allow explicitly for partial functions on a complete metric space.

Let us assume that we found some suitable complete metric space F of functions from states to states that is to provide meanings for programs. Let us now consider the program

$$\text{while true do } x := x \quad .$$

In the metric space approach the standard way to provide a meaning to such constructs is to define the operator

$$\Omega : F \to F$$

$$\Omega(f)(x) := \text{ if true}(x) \text{ then } f(x) \text{ else } x$$

and looking for the unique solution of

$$f = \Omega(f) \quad .$$

Such equations are typically solved [11] by showing that Ω obeys a contraction property, i.e. $\exists k < 1$ such that for all $f, g \in F$

$$d(\Omega(f), \Omega(g)) \leq k \cdot d(f, g) \quad .$$

Then one starts with a suitable element $f_0 \in F$, defines

$$f_i = \Omega(f_{i-1})$$

and obtains that $\lim f_i$ is a unique fixed point.

Unfortunately the above operator Ω does not satisfy a contraction property, as

$$d(\Omega(f), \Omega(g)) = d(f, g)$$

which holds independently of the metric definition. On the other hand in this particular case every function in F is a solution of $f = \Omega(f)$, but which should to be chosen as a meaning for our program?

More generally speaking, we are confronted with problems of assigning meaning to while-constructs, recursive programs etc. as the contraction property of the respective operators is not ensured. Had we used *cpo*'s instead and put for function f, g in a suitable function space

$$f \sqsubseteq g \iff f(x) = g(x) \vee f(x) = \perp$$

then for every while program the analogous operator Ψ displays the necessary continuity properties that guarantee the existence of a least solution ψ and the iteration

$$f_0 = \lambda x.\bot$$
$$f_i = \psi(f_{i-1}) \quad i \geq 1$$

yields $f_0 = f_1 = f_2 \ldots$ hence $f = \lambda x.\bot$ as desired solution.
Based on the above consideration we suspect that the metric space approach is not suitable for the definition of semantics that ignores intermediate states.

Problem 2

In a previous paper [19–21] we investigated the question under which conditions domain equations in the metric space approach are solvable. These equations yield the semantic domains for the interpretation of programs. The equations have the form

$$P = \mathcal{F}P$$

where P stands for a complete metric space and $\mathcal{F}$ is an endofunctor in the category CMS that has complete metric spaces as objects and non–distance–increasing functions as morphisms. A non–distance–increasing function from a metric space (X, d_X) to a metric space (Y, d_Y) is a function that satisfies $d_Y(f(x), f(y)) \leq d_X(x, y)$. See also [4] for a similar approach.
We established in [19–21] conditions each of which guarantees the existence and uniqueness of a solution for such equations. The basic idea in these conditions is to impose on the functor $\mathcal{F}$ a "contraction property". For this the concept of an "embedding" of a complete metric space (N, d_N) into a complete metric space (M, d_M) is used, which is a mapping that preserves distances. The two conditions established in [21] for the existence and uniqueness of a solution of $\mathcal{F}P = P$ are

i) There is a $k < 1$ such that for every embedding $e : N \to M$

$$d(\mathcal{F}N, \mathcal{F}M) \leq k \cdot d(N, M)$$

where $\mathcal{F}$ preserves embeddings. Here $d(N, M) = \sup_{x \in M} \inf_{y \in N} d\big(e(y), x\big)$.

ii) There is a $k < 1$ such that for every embedding $e : N \to M$ for which there is $c : M \to N$ satisfying $e \circ c = \mathrm{id}_N$ the following holds

$$\Big(\forall_{x \in M} d\big(x, e\,(c(x))\big) \leq \mu \quad \Rightarrow \quad \forall_{x \in \mathcal{F}(M)} d\Big(x, (\mathcal{F}e)\big((\mathcal{F}c)(x)\big)\Big) \leq k \cdot \mu \Big)$$

where $\mathcal{F}$ preserves embeddings.

In addition, we have shown in [21] that one equation treated in [11] does not have the solution proposed in [11], namely the equation

(II) $$P = \{p_0\} \cup \Big(A \to \wp_c\big((B \times P) \cup (C \to P)\big)\Big) \quad .$$

Here, P stands for a complete metric space with metric d, A, B, C are sets, $\wp_c(S)$ denotes the metric space of all closed subsets of S together with the Hausdorff metric. $C \to P$ denotes the metric space of functions from C to P endowed with the metric

$$d^*(f,g) = \sup_{c \in C} d\big(f(c), g(c)\big) \quad .$$

$B \times P$ is the metric space with metric

(III) $$d\big((a,x),(a',y)\big) = \begin{cases} \frac{1}{2}d(x,y) & \text{if } a = a' \\ 1 & \text{otherwise} \end{cases}$$

Clearly the functor of the equation

$$\mathcal{F}(X) = \{p_0\} \cup A \to \wp_c\big((B \times X) \cup (C \to X)\big)$$

does not exhibit any of the above contraction properties. This can be seen by observing that the functor

$$\mathcal{G}(X) = C \to X$$

that maps (X,d) to $(C \to X, d^*)$ neither satisfies i) nor ii) from above and that the functor $\wp_c$ cannot remedy the situation.

We do not know if this problematic equation has a solution at all. We only know that the standard approach to solve the equation by iteration as proposed in [11] does not work [21]. Examples of functors that involve the function space and do possess unique fixed points are e.g.

$$\mathcal{F}(X) = \{p_0\} \cup \big(A \to \wp_c(B \times X)\big)$$
$$\mathcal{F}(X) = \{p_0\} \cup \Big(A \to \wp_c\big(B \times \big(X \cup (C \to X)\big)\big)\Big)$$

In this context it is now interesting to observe that [2] and [4] use systematically a change of metric in the definition of functors. E.g. in [2] we find expressions as

$$\mathcal{F}(X) = \{p_0\} \cup^{\varepsilon} A \times X$$
$$\mathcal{F}(X) = \{p_0\} \cup^{\varepsilon} \wp_{cl}(A \times X)$$
$$\mathcal{F}(X) = \{p_0\} \cup^{\varepsilon} (A \to X)$$

where $0 < \varepsilon < 1$ and $\cup^{\varepsilon}$ means that the original metric of the operands of $\cup$ has to be multiplied by ε. Similarly in [4] the possibility of change of metric is given by providing a functor $\mathrm{id}^{\varepsilon}$.

The question that interests us here is, how a change of the definition of the metric for $C \to P$ influences the situation and how such an effect could be interpreted.

Let us, for example, redefine the metric for $C \to P$ to be

$$\tilde{d}(f,g) = \varepsilon \cdot \sup_{c \in C} d\big(f(c), g(c)\big)$$

for some $0 < \varepsilon < 1$.

Let us consider

$$\mathcal{G}'(X) = \{p_0\} \cup (C \to X)$$

where the metric on $C \to X$ is now $\tilde{d}$. It is easy to see that $\mathcal{G}'(X) = \{p_0\} \cup (C \to X)$ satisfies the above condition ii). As $\mathcal{F}(X) = A \times X$ satisfies ii) as well and $\wp_c$ preserves this property we conclude that

$$P = \{p_0\} \cup \Big(A \to \wp_c\big((B \times P) \cup (C \to P)\big)\Big) \quad .$$

has a unique solution, that is constructible via iteration starting from $\{p_0\}$. What have we done? We have changed the metric on the space $C \to X$ by multiplying with a constant < 1 which amounts to considering the functor that maps

$$(X, d) \vdash (C \to X, \tilde{d})$$

instead of

$$(X, d) \vdash (C \to X, d^*) \quad .$$

Clearly, $(C \to X, \tilde{d})$ and $(C \to X, d^*)$ are not isometrical.

How can this dependency on the metric definition be interpreted for programming language semantics? Does it make sense for the semantic definition of a programming language when we use $\tilde{d}$ instead of d^* in equations involving components of the form $C \to X$?

In trying to understand this situation we review the functor $\mathcal{F}(X) = \{p_0\} \cup (A \times X)$ with the metric described before. Let us consider the simple language

$$S \to \quad a \mid (S_1; S_2) \mid \text{loop}\, S$$

where $a \in A$ and where loop S means infinitely many iterations. To determine the meaning of programs of this language we use the unique complete metric space P that satisfies $P = \{p_0\} \cup (A \times P)$. The meaning function **Me** that maps statements to elements of P is given by

$$\begin{aligned} \mathbf{Me}(a) &= \langle a, p_0 \rangle \\ \mathbf{Me}(S_1; S_2) &= \mathbf{Me}(S_2) \circ \mathbf{Me}(S_1) \\ \mathbf{Me}(\text{loop}\, S) &= \lim p_i \qquad \text{where } p_0 = p_0,\ p_{i+1} = p_i \circ \mathbf{Me}(S) \end{aligned}$$

and the operation $\circ : P \to P$ is given by

$$\begin{aligned} p \circ p_0 &= p \\ p \circ \langle a, q \rangle &= \langle a, p \circ q \rangle \quad . \end{aligned}$$

For example the meaning of $(a_1; a_2)$ is

$$\begin{aligned} \mathbf{Me}(a_1; a_2) &= \mathbf{Me}(a_2) \circ \mathbf{Me}(a_1) \\ &= \langle a_2, p_0 \rangle \circ \langle a_1, p_o \rangle \\ &= \big\langle a_1, \langle a_2, p_0 \rangle \circ p_0 \big\rangle \\ &= \big\langle a_1, \langle a_2, p_0 \rangle \big\rangle \end{aligned}$$

and the meaning of loop a is

$$\mathbf{Me}(\text{loop } a) = \lim p_i$$

where

$$\begin{aligned} p_1 &= p_0 \circ \mathbf{Me}(a) \\ &= p_0 \circ \langle a, p_0 \rangle \\ &= \langle a, p_0 \rangle \\ p_2 &= \langle a, \langle a, p_0 \rangle \rangle \qquad \text{and so on.} \end{aligned}$$

By the metric definition of $A \times P$ (III) it is guaranteed that (p_i) is a Cauchy sequence in the complete metric space P and hence the meaning of loop a is welldefined.

In this very simple situation we can interprete the use of $P = \{p_0\} \cup (A \times P)$ and the metric on $A \times P$ as follows: the Cartesian product models the sequencing of actions. A program is "modelled" by an element $p \in P$ that reflects the actions p performs. Two programs S, S' are in distance $\left(\frac{1}{2}\right)^n$ if the first n actions of S and S' coincide and the $(n+1)$ st actions are different. If S and S' produce the same sequence of actions then they have distance 0.

In more complex languages that allow also for assignment and hence yield to the introduction of states the functor $\mathcal{F}(X) = A \times X$ with the above metric on $A \times X$ is used to formalize a notion of distance of processes based on the states or synchronization commands that they produce. For e.g. consider the language $\mathcal{L}_5$ of [11]

$$S \to x := s \mid \text{skip} \mid S_1; S_2 \mid S_1 \cup S_2 \mid S_1 \| S_2 \mid S^* \mid C \mid \overline{C} \mid S \setminus_1 C \mid S \setminus_2 C \mid \Delta$$

with its associated domain equation

$$P = \{p_0\} \cup \Big(\Sigma \to \wp_c\big((\Sigma \cup \Gamma) \times P\big)\Big) \quad .$$

Here the meaning of a program is either p_0 or a function that maps states in Σ to closed subsets of $(\Sigma \cup \Gamma) \times P$, i.e.

$$\begin{aligned} \mathbf{Me} : \text{programs} &\to P \\ S &\mapsto \lambda\sigma.\{\langle \beta, p \rangle \ldots\} \end{aligned}$$

where $\beta \in \Sigma$ is either the new state reached or the meaning of a synchronization command and p describes the "remainder" of the computation. Again, the definition of the metric d for $A \times X$ ensures that programs that produce the same subsequent states or commands for any arbitrary state σ are "closer" than those that differ.

Example

$$S_1 \equiv x := 0; y := x + 1$$

$$S_2 \equiv x := 0; y := x + 2$$

$$S_3 \equiv x := 1; y := x + 2$$

$$p_1 = \mathbf{Me}(S_1) = \lambda\sigma.\left\{\left\langle \sigma_x^0, \lambda\overline{\sigma}.\left\{\langle \overline{\sigma}_y^{\mathcal{V}(x+1)\overline{\sigma}}, p_0\rangle\right\}\right\rangle\right\}$$

$$p_2 = \mathbf{Me}(S_2) = \lambda\sigma.\left\{\left\langle \sigma_x^0, \lambda\overline{\sigma}.\left\{\langle \overline{\sigma}_y^{\mathcal{V}(x+2)\overline{\sigma}}, p_0\rangle\right\}\right\rangle\right\}$$

$$p_3 = \mathbf{Me}(S_3) = \lambda\sigma.\left\{\left\langle \sigma_x^1, \lambda\overline{\sigma}.\left\{\langle \overline{\sigma}_y^{\mathcal{V}(x+2)\overline{\sigma}}, p_0\rangle\right\}\right\rangle\right\}$$

$$d(p_1, p_2) = \frac{1}{2} \qquad d(p_1, p_3) = 1 \quad .$$

Let us now consider the role of the function space construction $\mathcal{G}(X) = C \to X$. In practical applications this functor is used as in the above example and e.g. in the form

$$(\alpha) \qquad \mathcal{F}(P) = \{p_0\} \cup \left(\Sigma \to \wp_c(\Sigma \times P)\right)$$

where Σ is again the set of states [11], or in the form

$$(\beta) \qquad \mathcal{F}(P) = \{p_0\} \cup \left(\Sigma \to \wp_c\Big((\Sigma \cup \Gamma) \times \big(P \cup (V \times P) \cup (V \to P)\big)\Big)\right)$$

where V is the set of values [11], or as

$$(\gamma) \qquad \mathcal{F}(P) = \{p_0\} \cup \wp_c\Big((\Gamma \cup V \cup \{\varepsilon\}) \times \big(P \cup (V \to P)\big)\Big)$$

[11], or

$$(\delta) \qquad \mathcal{F}(P) = \{p_0\} \cup^{\frac{1}{2}} \left(\Sigma \to \wp_c(\mathrm{Step}_P)\right)$$

where

$$\mathrm{Step}_P = (\Sigma \times P) \cup \mathrm{Send}_P \cup \mathrm{Answer}_P$$

$$\mathrm{Send}_P = \mathrm{Obj} \times \mathrm{MName} \times \mathrm{Obj}^* \times (\mathrm{Obj} \to P)$$

$$\mathrm{Answer}_P = \mathrm{Obj} \times \mathrm{MName} \times \big(\mathrm{Obj}^* \to (\mathrm{Obj} \to P) \to P\big)$$

[2].

In (α) the function space construction serves to describe the dependency of the process behaviour on the states. A process p is either p_0 or a function that – depending on the state – produces a new state followed by a "remainder" process q. The distance of two processes is measured by looking at the produced states.

In (β) the same remark holds for the first instance of the function space functor. In addition, after having produced a state or a communication command a remainder process q starts or a remainder process q is selected depending on a received value. Similarly the distance of two processes is measured – roughly speaking – by the states, respectively communication commands, they produced.

One can see that the same observations hold for equations of type (γ) or (δ).

Processes always produce some state or message and continue with a "remainder" process and the whole situation may depend on the state or some received value. The distance of processes should

be measured comparing their respective actions. The function space construction serves to describe the fact that the behaviour of a process may depend on e.g. states or values that are received via message. In the light of the above discussion we can see no reason whatsoever why – in the context of programming language semantics – one should consider the functor $\mathcal{G}'(X) = (C \to X, \tilde{d})$ instead of $\mathcal{G}(X) = (C \to X, d^*)$. Also, we could not think of a reasonable language for which the necessity of using $\mathcal{G}'$ instead of $\mathcal{G}$ arises. In particular we cannot think of any reasonable language that has

$$P = \{p_0\} \cup (C \to P)$$

as its domain equation. All examples of languages treated in [2, 3, 11, 18, 32] possess corresponding process domain equations that are solvable without change of metric!

Problem 3

In the *cpo*–approach for the semantic definition of programming languages it is common to find domain equations of the form

$$D = A \cup [D \to D]$$

[39], i.e. function space constructions where the domain in question occurs on the left hand side of the function space functor.

Let us assume that we want to allow for a similar construction in the metric space approach, i.e. we want to define a functor $\mathcal{H}$ in **CMS** that maps complete metric spaces X to some function space $X \to \ldots$ and deals with the arrows appropriately. How could this be done? It is obvious to define various mappings on the objects of **CMS**

$$\mathcal{H}_1(X) = X \to X$$

$$\mathcal{H}_2(X) = X \to X_0 \qquad \text{for some fixed } X_0$$

etc. It is not so clear, however, how to treat the arrow of **CMS**. Let e.g. N, M be complete metric spaces, $e : N \to M$ an embedding, $c : M \to N$ non–distance–increasing function. How should we define

$$\mathcal{H}_1(e) : (N \to N) \longrightarrow (M \to M) \quad ?$$

If f is a function from N to N how do we make from it a function from M to M? It is this problem that led [4] to work with a different category of complete metric spaces than we did above. [4] consider a category **C** that contains as objects nonempty complete metric spaces and as arrows pairs $\iota = (i, j)$ of non–distance–increasing functions such that $i : N \to M$ is an embedding, i.e. preserves distances, j maps M to N and $j(i(x)) = x \quad \forall x \in N$. In this category one can easily define functors that map an object X to some function space $X \to \ldots$. The question, however, arises if there is the necessity to consider functors like this when using the metric space approach for the semantic specification of programming languages. [4] argue that the necessity for "environments" or "continuations" calls for such functors and name the work [2] as an example, where such situations occur.

We checked this argument by looking into all available specifications of languages using the metric space approach including [2]. We found indeed that there are definitions of environments Γ like

$$\Gamma = \mathrm{Stm}\, V \to \left(I \to (P \overset{NDI}{\longrightarrow} P)\right)$$

see e.g. [3, page 44] but in this case P was constructed before as the solution of

$$P = \{p_0\} \cup \left(\Sigma \to \wp_c(A \times P)\right)$$

i.e. for that part of the construction that needs the concept of a functor, namely the solution of the fixed point equation, P does not occur on the left side of the functions space functor. For the definition of Γ we do not need the concept of a functor at all. The same situation is also found in [3, page 56] where $\Gamma = \Gamma_1 \times \Gamma_2$ and

$$\Gamma_1 = \mathrm{Stm}\, V \to (AO \to P \overset{NDI}{\longrightarrow} P)$$

$$\Gamma_2 = \mathrm{CNam} \to (AO \to P)$$

where again P is constructed beforehand as solution of

$$P = \{p_0\} \cup \left(\Sigma \to \wp_c(A \times P)\right) \quad .$$

Similarly in [2, page 18ff] a meaning function is defined as

$$[\ \ ]_S : \mathrm{Stat} \to \mathrm{Env} \to \mathrm{AObj} \to \mathrm{Cont}_S \to P \qquad \text{where}$$

$\mathrm{Cont}_S = P$ {continuations}

$\mathrm{Env} = (\mathrm{Name} \to \mathrm{AObj} \to P) \times (\mathrm{MName} \to \mathrm{CName} \to \mathrm{AObj} \to \mathrm{Obj}^* \to (\mathrm{Obj} \to P) \to P)$

{environments}

and P is previously defined by

$$P = \{p_0\} \cup^{\frac{1}{2}} \left(\Sigma \to \wp_c(\mathrm{Step})\right)$$

see (δ).

So we suppose that neither environment nor continuation do need a function space construction $\mathcal{F}(X) = X \to \ldots$. As far as we can see an equation involving $\mathcal{F}(X) = X \to \ldots$ might occur, if we have to deal with a class C of programming constructs (e.g. procedures) that can be "applied" to objects of the same class C.

Conclusion

We discussed three problems that arise in the context of denotational semantics of imperative language. The first is concerned with the question of how "abstract" the metric space approach is. We argue that it is less abstract than the *cpo* approach. The second problem deals with the relevance of the choice of metric in the metric space approach. Here we claim that the Cartesian product is the relevant operator for determining the distance whereas $\wp_c$ is responsible for nondeterminism and the function space construction describes dependency on states and / or messages. The third problem deals with the need for functors of the type $\mathcal{F}(X) = X \to \ldots$. We think that the above considerations lead to a better understanding of the metric space approach and its relation to the *cpo* –approach.

References

1. Abrahamson, K.: Modal logic of concurrent nondeterministic programs. Lecture notes in C. S. 70, 21-34. Springer (1979)

2. America, P., de Bakker, J. W., Kok, J. N., Rutten, J. J. M. M.: A denotational semantics of a parallel object–oriented language. Report CS-R8626. Centre for Mathematics and Computer Science Amsterdam (1986).

3. America, P., de Bakker, J. W.: Designing equivalent semantic models for process creation. Report CS-R8732. Centre for Mathematics and Computer Science Amsterdam (1987).

4. America, P., Rutten, J. J. M. M.: Solving reflexive domain equations in a category of complete metric spaces. Report CS-R8709. Centre for Mathematics and Computer Science Amsterdam (1987). Also in: 3rd Workshop on Mathematical Foundations of Programming Language Semantics. Springer Lecture Notes in Computer Science 298 (1988)

5. Apt, K. R.: Recursive assertions of parallel programs. Acta Informatica 15, 219-232 (1981)

6. Apt, K. R., Francez, N., De Roever, W. P.: A proof system for communicating sequential processes. ACM TOPLAS 2, 359-385 (1980)

7. Cousot, P., Cousot, R.: Semantic analysis of communicating sequential processes. Lecture Notes in Computer Science 85, 119-133 (1980)

8. De Bakker, J. W., Meyer, J., Olderog, E., Zucker, J.: Transition systems, infinitary languages and the semantics of uniform concurrency. Pro. 17th ACM Symposium on Theory of Computing (1985)

9. De Bakker, J. W., Kok, J. N., Meyer, J.–J. Ch., Olderog, E., Zucker, J.: Contrasting themes in the semantics of imperative concurrency. Lecture Notes in Computer Science 224 (1986)

10. De Bakker, J. W., Meyer, J.–J. Ch.: Metric semantics for concurrency. Report CS-R8803. Centre for Mathematics and Computer Science Amsterdam (1988).

11. De Bakker, J.W., Zucker, J.: Processes and the denotational semantics of concurrency. Information and Control 54, 70-120 (1982)

12. Francez, N., Hoare, C. A. R., Lehman, D., De Roever, W. P.: Semantics of nondeterminism, concurrency and communication. J. Comp. Sys. Sci. 19, 290-308 (1979)

13. Golson, W. G., Rounds, W. C.: Connections between two theories of concurrency. Information and Control 57, 102-124 (1983)

14. Hahn, H: Reelle Funktionen. Chelsea, New York (1948)

15. Hailpern, B. T.: Verifying concurrent processes using temporal logic. Lecture Notes in Computer Science 129 (1982)

16. Hoare, C. A. R.: Communicating sequential processes. Comm. ACM 21, 666-677 (1978)

17. Akilov, G., Kantorovic, L.: Functional analysis. Pergamon (1982)

18. Majster-Cederbaum, M. E., Zetzsche, F.: The relation between Hoare-like and de Bakker-style semantics for concurrent languages. Draft paper.

19. Majster-Cederbaum, M. E.: On the uniqueness of fixed points of endofunctors in a category of complete metric spaces. Information processing letters 29 (1988)

20. Majster-Cederbaum, M. E.: The contraction property is sufficient to guarantee the uniqueness of fixed points of endofunctors in a category of complete metric spaces. To appear in Information processing letters (1989)

21. Majster-Cederbaum, M. E., Zetzsche, F.: Foundation for semantics in complete metric spaces. Technical report 1/1987 Reihe Informatik, Fakultät für Mathematik und Informatik der Universität Mannheim (1987). Also accepted for publication in Information and Computation.

22. Milner, R.: A calculus for communicating systems. Lecture Notes in Computer Science 92 (1980)

23. Olderog, E. R.: Process Theory. Semantics, specification and verification. Lecture Notes in Computer Science 224 (1986)

24. Olderog, E. R., Hoare, C. A. R.: Specification oriented semantics for communicating processes. Acta Informatica 23, 9-66 (1986)

25. Owicki, S., Gries, D.: Verifying properties of parallel programs: an axiomatic approach. Comm. of the ACM 19, 279-285 (1976)

26. Park, D.: On the semantics of fair parallelism. Lecture Notes in Computer Science 86, 504-526 (1980)

27. Plotkin, G. D.: A power domain construction. SIAM J. Computing 5, No (1976)

28. Plotkin, G. D.: An operational semantics for CSP. IFIP Conference on Formal Description of Programming Concepts. North Holland. Ed. D. Bjorner (1983)

29. Pnueli, A.: A temporal logic for concurrent programs. Lecture Notes in Computer Science 70 (1979)

30. Reynolds, J. C.: Notes on a lattice-theoretic approach to the theory of computation. System and Information Science Department Syracuse University (1972)

31. Rutten, J. J. M. M., Zucker, J. I.: A semantic approach to fairness. Report CS-R8759. Centre for Mathematics and Computer Science Amsterdam (1987)

32. Schmitt, A.: A de Bakker style semantics for dynamic process creation. Technical Report 04/1988 Fachbereich 10 Universität Saarbrücken (1988)

33. Scott, D. S.: Continuous lattices, toposes, algebraic geometry and logic. Proc. 1971 Dalhousie Conference. Lecture Notes in Math. 274, Springer 97-136, (1972)

34. Scott, D. S.: Data types as lattices. SIAM J. of Computing, 5 (1976)

35. Schwartz, J.: Denotational semantics of parallelism. Lecture Notes in Computer Science 70 (1979)

36. Smyth, M. B.: Power domains. J. Comp. Sys. Sciences 16 (1978)

37. Smyth, M. B.: Quasi Uniformities: Reconciling domains with metric spaces. Manuscript.

38. Smyth, M. B., Plotkin, G. D.: The category-theoretic solution of recursive domain equations. SIAM J. of Computing 11, 761-781 (1982)

39. Stoy, J.: Denotational semantics of programming languages: The Scott–Strachey approach. MIT Press, Cambridge, Mass. (1977)

40. Wand, M.: Fixed-point constructions in order-enriched categories. Theor. Comp. Sci. 8, 13-30 (1979)

41. Weihrauch, K., Schreiber, U.: Embedding metric spaces into *cpo*. Manuscript.