A POWERDOMAIN FOR COUNTABLE NON-DETERMINISM
(Extended Abstract)

G.D. Plotkin
Dept. of Computer Science
University of Edinburgh

1. Introduction

This paper proposes a general powerdomain for countable nondeterminism and uses it to give the denotational semantics of a simple imperative programming language with a fair parallel construct. As already known from the simple case of a discrete cpo [AP] countable nondeterminism seems to force the consideration of non-continuous functions. In the classical Scott-Strachey approach only continuous functions are allowed and it is necessary to extend the mathematics to a weaker kind of continuity and show how it is still possible to specify and work with least solutions to recursive equations for elements of domains and initial solutions to recursive domain equations.

Fairness or the finite delay property is a natural assumption that has been studied in many settings by many authors. The general idea is that no subprocess is to be delayed indefinitely. More exactly there are two main ways to define a fair computation sequence:

Weak Fairness No event is almost always possible (unless the sequence is finite).

Strong Fairness No event is infinitely often possible.

These statements are deliberately informal: all depends on what counts as an event (see also [AO,Kwo,LPS,Man,Par]). In the present paper only weak fairness is studied as there are no possible strong fairness phenomena in the simple language at hand.

Section 2 begins by defining an operational semantics for our language. This provides a concrete model against which it proves possible to test any denotational semantics. The definition is of the well-known *restrictive* or *negative* kind implied by the above formulations of fairness: first specify all execution sequences and then restrict attention to the fair ones (= rule out the unfair ones). Since our language is richer than the usual case of n sequential processes with shared memory the techniques used may be of interest. They comprise a *structural operational semantics* [Plo2] to specify transitions, *redexes* (here called *actions*) to specify potential occurrences (in our case these are also all possible) and *residuals* to trace potential occurrences through transitions [Bar]. Now it is well-known that fairness (in either form) implies countable nondeterminism. Section 2 concludes by using this idea on the meta-level to provide a *generative* or *positive* operational semantics in which all computation sequences are fair (and which gives all the fair sequences that the restrictive semantics does); this is proved in Theorem 1.

Section 3 begins with a review of the discrete case which suggests a suitable form of weak continuity (= ω_1-continuity $\overset{def}{=}$ preservation of lubs of increasing ω_1-sequences) and a suitable form of cpo (having a $\perp$ and lubs of ω_0- and ω_1-sequences). These assumptions permit least fixed-points to exist and give rise to a form of Scott induction (called ω_1-induction) that is used extensively in Section 4. The essential feature for handling countable nondeterminism seems to be the ability to take arbitrary countable unions. Now in the case of bounded (= finite) nondeterminism one needed only to take finite unions; the abstract view is that *semilattices* were needed and in [HP] all the various powerdomains previously considered were characterised as suitable free continuous semilattices. Here *σ-semilattices* seem indicated (as noted independently by Axel Poigné) and several candidates for the free weakly-continuous σ-semilattice are shown to exist (Theorem 2). Now the lack of continuity extends also to the powerdomain construction itself and that makes it impossible to solve recursive domain equations by the usual categorical analogue of the formula for the least fixed-point of a continuous function. In Theorems 5 and 6 and Corollary 1 an extension of the work in [SP] is presented that allows such equations to be solved in the presence of weak continuity (and Theorem 5 appears already in [AK]).

Section 4 begins with an attempt to use the preferred candidate for the powerdomain construct to give a denotational semantics to the example language. The idea is to use a recursively-specified domain of resumptions (as in [Plo1]). To the author's surprise, however, this does not work as it does not seem possible to define the semantics of the parallel construct; the problem is that with the preferred candidate there remains some continuity requirements and these are violated. However these difficulties do not arise with the alternate candidate. Finally various relationships between the operational and denotational semantics are established. Theorem 7 shows that the operational semantics determines the denotational semantics, and Theorem 8 shows the converse for some simple notions of behaviour derived from the operational semantics.

Clearly there remains much to do. The proposed powerdomains are shown to exist by highly nonconstructive methods of category theory. Direct existence along the lines of [Plo1,Smy] should be established and an investigation made of the effectiveness of the constructions and functions involved. This is extremely important as the loss of continuity seems to violate Scott's most reasonable thesis that all computable functions are continuous. Next the relation between the various semantics needs further investigation (see [HP] for some discussion of the so-called full-abstraction issue). The successful employment of ω_1-induction encourages an attempt to use it as a means of proving correct the many classical algorithms based on underlying fairness assumptions. It also seems feasible to extend the work to extensions of the current language where, in particular, both weak and strong fairness can be considered. Finally it is not at all clear what can be done in other settings where fairness considerations arise such as languages for message-passing or communication or dataflow languages where there is the difficult "fair merge" problem.

2. Operational Semantics

By adding a parallel construct to a simple imperative language we obtain a first setting for studying fairness. The language has three syntactic categories.

1. ACom A given set of atomic commands, ranged over by ac.
2. BExp A given set of Boolean expressions ranged over by b.
3. Com A set of commands ranged over by c and with abstract syntax given by: $c ::= ac \mid \underline{\text{skip}} \mid c_1;c_2 \mid \underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2 \mid \underline{\text{while}}\ b\ \underline{\text{do}}\ c \mid c_1 || c_2$.

Operational semantics is provided via a labelled transition relation [Kel,Mil] on a set, Γ, of configurations (ranged over by γ); To define Γ assume a given denumerable set S of states (ranged over by σ). Then $\Gamma \overset{\text{def}}{=} \{\langle c,\sigma\rangle\} \cup \{\sigma\}$. We will specify a transition relation $\rightarrow\ \subseteq \Gamma \times A \times \Gamma$ where $A \overset{\text{def}}{=} \{1,2\}^*$ is the set of actions (ranged over by a and b). The idea is that in a relation $\gamma \xrightarrow{a} \gamma'$ the action indicates which of the possible transitions is taken. We assume that the semantics of atomic commands and Boolean expressions are given by functions $\mathcal{A}$: ACom -> (S -> S) and $\mathcal{B}$: BExp -> (S -> T) (where T = {tt,ff} is the set of truth-values). Now the following rules specify the transition relation by structural induction on commands [Plo2].

Atomic Commands $\langle ac,\sigma\rangle \xrightarrow{\varepsilon} \mathcal{A}[\![ac]\!](\sigma)$

Skip $\langle \underline{\text{skip}},\sigma\rangle \xrightarrow{\varepsilon} \sigma$

Composition
$$\frac{\langle c_1,\sigma\rangle \xrightarrow{a} \langle c_1',\sigma'\rangle \mid \sigma'}{\langle c_1;c_2,\sigma\rangle \xrightarrow{a} \langle c_1';c_2,\sigma'\rangle \mid \langle c_2,\sigma'\rangle}$$

Conditional

1. $\langle \underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2,\sigma\rangle \xrightarrow{\varepsilon} \langle c_1,\sigma\rangle$ (if $\mathcal{B}[\![b]\!](\sigma) = tt$)
2. $\langle \underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2,\sigma\rangle \xrightarrow{\varepsilon} \langle c_2,\sigma\rangle$ (if $\mathcal{B}[\![b]\!](\sigma) = ff$)

Repetition

1. $\langle \underline{\text{while}}\ b\ \underline{\text{do}}\ c,\sigma\rangle \xrightarrow{\varepsilon} \langle c;\underline{\text{while}}\ b\ \underline{\text{do}}\ c,\sigma\rangle$ (if $\mathcal{B}[\![b]\!](\sigma) = tt$)
2. $\langle \underline{\text{while}}\ b\ \underline{\text{do}}\ c,\sigma\rangle \xrightarrow{\varepsilon} \sigma$ (if $\mathcal{B}[\![b]\!](\sigma) = ff$)

Parallel

1. $$\frac{\langle c_1,\sigma\rangle \xrightarrow{a} \langle c_1',\sigma'\rangle \mid \sigma'}{\langle c_1||c_2,\sigma\rangle \xrightarrow{1a} \langle c_1'||c_2,\sigma'\rangle \mid \langle c_2,\sigma'\rangle}$$

$$2.\quad \frac{\langle c_2,\sigma\rangle \xrightarrow{a} \langle c_2',\sigma'\rangle \mid \sigma'}{\langle c_1 \| c_2,\sigma\rangle \xrightarrow{2a} \langle c_1 \| c_2',\sigma'\rangle \mid \langle c_1,\sigma'\rangle}$$

To see how the actions are working define a function, Act, sending commands to non-empty finite subsets of A by: $\mathrm{Act}(ac) = \mathrm{Act}(\mathbf{skip}) = \mathrm{Act}(\mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2) = \mathrm{Act}(\mathbf{while}\ b\ \mathbf{do}\ c) = \{\varepsilon\}$; $\mathrm{Act}(c_1;c_2) = \mathrm{Act}(c_1)$; $\mathrm{Act}(c_1 \| c_2) = 1\,\mathrm{Act}(c_1) \cup 2\,\mathrm{Act}(c_2)$ and extend it to configurations by putting $\mathrm{Act}(\langle c,\sigma\rangle) = \mathrm{Act}(c)$; $\mathrm{Act}(\sigma) = \emptyset$. One can think of $\mathrm{Act}(\gamma)$ as the set of potential events of γ.

Lemma 1 1. $\forall\gamma\forall a\forall\gamma'.\gamma \xrightarrow{a} \gamma' \supset a \in \mathrm{Act}(\gamma)$

2. $\forall\gamma\forall a \in \mathrm{Act}(\gamma)\exists!\gamma'.\gamma \xrightarrow{a} \gamma'$

3. $\forall\gamma\forall a(\exists\sigma'\ \gamma \xrightarrow{a} \sigma') \equiv \{a\} = \{\varepsilon\} = \mathrm{Act}(\gamma)$

Intuitively parts 1 and 2 of Lemma 1 say there is a 1-1 correspondence between potential and possible events. To be able to express fairness we now need to see how possible actions change from one transition to another. For any a,b in $\mathrm{Act}(\gamma)$ we define the residual actions $\mathrm{Res}(b,\gamma,a) \subseteq A$ of b after the a transition from γ by induction on the command in γ.

$\mathrm{Res}(b,\langle c,\sigma\rangle,a) = \emptyset$ (if c is atomic, skip, a conditional or a repetition)

$\mathrm{Res}(b,\langle c_1;c_2,\sigma\rangle,a) = \mathrm{Res}(b,\langle c_1,\sigma\rangle,a)$

$$\mathrm{Res}(1b_1,\langle c_1 \| c_2,\sigma\rangle,a) = \begin{cases} 1\,\mathrm{Res}(b_1,\langle c_1,\sigma\rangle,a_1) & (\text{if } a=1a_1 \text{ and } \langle c_1,\sigma\rangle \xrightarrow{a_1} \langle c_1',\sigma'\rangle) \\ \emptyset & (\text{if } a=1a_1 \text{ and } \langle c_1,\sigma\rangle \xrightarrow{a_1} \sigma') \\ \{1b_1\} & (\text{if } a=2a_2 \text{ and } \langle c_2,\sigma\rangle \xrightarrow{a_2} \langle c_2',\sigma'\rangle) \\ \{b_1\} & (\text{if } a=2a_2 \text{ and } \langle c_2,\sigma\rangle \xrightarrow{a_2} \sigma') \end{cases}$$

$\mathrm{Res}(2b_2,\langle c_1 \| c_2,\sigma\rangle,a)$ is defined symmetrically

Lemma 2 1. Either $\mathrm{Res}(b,\gamma,a)$ is empty and $b = a$ or else it is a singleton $\{b_1b_2\}$ (where $b = b_1ib_2$ for some i in $\{1,2,\varepsilon\}$) and $b \neq a$.

2. If $\gamma \xrightarrow{a} \gamma'$ and $b' \in \mathrm{Res}(b,\gamma,a)$ then $b' \in \mathrm{Act}(\gamma')$.

Definition 1 An execution sequence $\gamma = \gamma_0 \xrightarrow{a_0} \gamma_1 \xrightarrow{a_1} \ldots \to \gamma_n \xrightarrow{a_n} \ldots$ of γ is unfair if it is infinite and there is an infinite sequence $b_m, b_{m+1}, \ldots$ where for every $k \geq m$ $b_k \in \mathrm{Act}(\gamma_k)$ and $b_{k+1} \in \mathrm{Res}(b_k,\gamma_k,a_k)$.

Pictorially an unfair sequence looks like this

$$\begin{tikzcd}
\gamma_0 \arrow[r, "a_0"] & \cdots \arrow[r] & \gamma_m \arrow[r, "a_m"] \arrow[d, dash, "\mathrm{Act}"'] & \gamma_{m+1} \arrow[r] \arrow[d, dash, "\mathrm{Act}"'] & \cdots \arrow[r] & \gamma_k \arrow[r, "a_k"] \arrow[d, dash, "\mathrm{Act}"'] & \gamma_{k+1} \arrow[r] \arrow[d, dash, "\mathrm{Act}"'] & \cdots \\
 & & b_m \arrow[r, dash, "\mathrm{Res}"'] & b_{m+1} \arrow[r, dash] & \cdots \arrow[r, dash] & b_k \arrow[r, dash, "\mathrm{Res}"'] & b_{k+1} \arrow[r, dash] & \cdots
\end{tikzcd}$$

Intuitively the b_k correspond to an event which is almost always possible but never actual.

Definition 2 A configuration diverges if it has an infinite fair execution sequence.

When commands are run for their final state a suitable measure of their behaviour is given by the relational approach modified to deal with termination. For any command c we define its relation and its termination domain by

$R[\![c]\!] = \{\langle\sigma,\sigma'\rangle \mid \langle c,\sigma\rangle \to^* \sigma'\}$ (where $\to\ = \bigcup\{\xrightarrow{a} \mid a \in A\}$) and $T[\![c]\!] = \{\sigma \mid \langle c,\sigma\rangle$ converges$\}$ respectively.

Generative Semantics

The operational semantics presented above can be considered negative or restrictive in that first a set of execution sequences is considered and then certain ones are ruled out as unfair. Now a positive or generative operational semantics is proposed in which only (and all) fair sequences can be generated in the first place. The idea is that at any point in a fair execution of $c_1 || c_2$ there is an upper bound on the number of transitions that c_1 makes before c_2 makes one, since otherwise there is an action of c_2 almost always possible but never taken (and similarly for c_2).

To formalise the idea we add constructs $c_1 ||^m c_2$ and $c_1 ||_m c_2$ (for $m \geq 0$) to the language giving a new set gCom of commands. To execute $c_1 ||^m c_2$ one executes m+1 steps of c_1 (unless prevented by the termination of c_1); and then executes $c_1 ||_n c_2$ for an arbitrary $n \geq 0$; the execution $c_1 ||_m c_2$ proceeds symmetrically. As before, the generative semantics is given by a transition relation $\rightarrow_g \subseteq \Gamma_g \times A \times \Gamma_g$ where (evidently) $\Gamma_g = (\text{gCom} \times S) \cup S$; the rules are the same as before except for the parallel construct and ones for the new constructs.

<u>Parallel</u> 1. $$\frac{\langle c_1 ||^m c_2, \sigma\rangle \overset{a}{\rightarrow}_g \gamma}{\langle c_1 || c_2, \sigma\rangle \overset{a}{\rightarrow}_g \gamma} \quad (m \geq 0) \qquad 2. \frac{\langle c_1 ||_m c_2, \sigma\rangle \overset{a}{\rightarrow}_g \gamma}{\langle c_1 || c_2, \sigma\rangle \overset{a}{\rightarrow}_g \gamma} \quad (m \geq 0)$$

<u>Left-Parallel</u> 1. $$\frac{\langle c_1, \sigma\rangle \overset{a}{\rightarrow}_g \langle c_1', \sigma'\rangle | \sigma'}{\langle c_1 ||^0 c_2, \sigma\rangle \overset{1a}{\rightarrow}_g \langle c_1' ||_n c_2, \sigma'\rangle | \langle c_2, \sigma'\rangle} \quad (n \geq 0)$$

2. $$\frac{\langle c_1, \sigma\rangle \overset{a}{\rightarrow}_g \langle c_1', \sigma'\rangle | \sigma'}{\langle c_1 ||^{m+1} c_2, \sigma\rangle \overset{1a}{\rightarrow}_g \langle c_1' ||^m c_2, \sigma'\rangle | \langle c_2, \sigma'\rangle} \quad (m \geq 0)$$

<u>Right-Parallel</u> (Symmetric to Left-Parallel)

To connect up the two approaches let w: $\Gamma_g \rightarrow \Gamma$ be the function which removes the labels of constituent parallel commands

<u>Lemma 3</u> If $\gamma \overset{a}{\rightarrow}_g \gamma'$ then $w(\gamma) \overset{a}{\rightarrow} w(\gamma')$

Now we can state a theorem that insofar as execution sequences are concerned the generative semantics captures the restrictive semantics.

<u>Theorem 1</u> For any execution sequence $\gamma_1 \overset{a_1}{\rightarrow}_g \gamma_2 \overset{a_2}{\rightarrow} \ldots$ the execution sequence $w(\gamma_1) \overset{a_1}{\rightarrow} w(\gamma_2) \overset{a_2}{\rightarrow} \ldots$ is fair and every fair execution sequence can be found thus.

3. <u>Powerdomains</u>

If we are to give denotational semantics to our language with its fair parallel construct then we need to be able to solve recursive domain equations involving a powerdomain for countable nondeterminism; for this purpose we want a powerdomain functor over a suitable category of partial orders. We start with a review of the discrete case.

<u>Definition 3</u> For any countable set X the powerdomain $\mathcal{E}(X_\perp)$ is the set of non-empty subsets of $X_\perp$ under the Egli-Milner partial order

$$X \sqsubseteq Y \text{ iff } (\forall x \in X \exists y \in Y. x \sqsubseteq y) \wedge (\forall y \in Y \exists x \in X. x \sqsubseteq y)$$

The <u>singleton function</u> $\{\cdot\}: X_\perp \rightarrow \mathcal{E}(X_\perp)$ and the <u>subset relation</u>, $\subseteq$, on $\mathcal{E}(X_\perp)$ have the usual set-theoretic definitions.

<u>Fact 1</u> 1. The powerdomain $\mathcal{E}(X_\perp)$ has a least element $\{\perp\}$, lubs of increasing ω_0-chains and increasing ω_1-chains (the latter being eventually constant).

2. Binary union $\cup$: $\mathcal{E}(X_\perp)^2 \rightarrow \mathcal{E}(X_\perp)$ is ω_0-and and ω_1-continuous and countable union $\bigcup$: $\mathcal{E}(X_\perp)^\omega \rightarrow \mathcal{E}(X_\perp)$ is ω_1-continuous but <u>not</u> in general ω_0-continuous.

3. For every monotonic f: $X_\perp \rightarrow \mathcal{E}(Y_\perp)$ (where Y is also any countable set) there is a unique function $f^\dagger$: $\mathcal{E}(X_\perp) \rightarrow \mathcal{E}(Y_\perp)$ such that the following diagram commutes

$$\begin{tikzcd}
X_\perp \arrow[d, "\{\cdot\}"'] \arrow[dr, "f"] & \\
\mathcal{E}(X_\perp) \arrow[r, "f^\dagger"] & \mathcal{E}(Y_\perp)
\end{tikzcd}$$

and such that $f^\dagger$ is ω_0-and ω_1-continuous (wrt $\sqsubseteq$) and preserves countable unions. Also if f is strict so is $f^\dagger$.

4. As a function, $(\cdot)^\dagger$ is monotonic, ω_1-continuous but not in general ω_0-continuous.

The non-continuity of extension leads to the non-continuity of important functionals for which a guaranteed fixed-point is required. Luckily we are saved by the completeness of the spaces involved.

<u>Fact 2</u> Let D be a po with a $\perp$ and lubs of increasing ω_0-and ω_1-sequences. Then any ω_1-continuous function f: D -> D has a least fixed-point $\mathrm{Fix}_f \overset{\mathrm{def}}{=} f^{\omega_1}$ where for $\kappa<\omega_1$ the κth iterate f^κ is defined by

$$f^0 = \perp,\ f^{\kappa+1} = f(f^\kappa),\ f^\lambda = \bigsqcup_{\kappa<\lambda} f^\kappa \ (\lambda \text{ a limit ordinal}).$$

How are we to react in the light of the above (carefully selected!) experience? In general it seems that we want a countable union function and that will involve us in non-ω_0-continuous (but ω_1-continuous) functions. On the other hand the partial orders we will use can be expected to be not too bad having lubs of increasing ω_0-and ω_1-sequences.

<u>Definition 4</u> Let **Pos** $(\kappa,\ldots;\lambda,\ldots)$ be the category whose objects are partial orders with lubs of increasing κ-chains and ... and whose morphisms are those monotonic functions preserving lubs of increasing λ-chains and ... . In particular set

$$\mathbf{A} = \mathbf{Pos}\,(\omega_0,\omega_1;\omega_0,\omega_1) \qquad \mathbf{A}_1 = \mathbf{Pos}\,(\omega_0,\omega_1;\omega_1)$$

Here $\mathbf{A}$ is the nicest category of partial orders we could hope to work in, but $\mathbf{A}_1$ is the expected one. Both are Cartesian closed with the usual Cartesian product and pointwise-ordered function spaces (written D -> E and D $->_1$ E in $\mathbf{A}$, $\mathbf{A}_1$ respectively). In $\mathbf{A}_1$ the least fixed-point operator Fix: (D $->_1$ D) $->_1$ D is ω_1-continuous but not, in general, ω_0-continuous.

In [HP] the available powerdomains for bounded nondeterminism [Plo1,Smy] were characterised as free semilattices over a category of partial orders. It now seems appropriate to try free σ-semilattices.

<u>Definition 5</u> A <u>semilattice</u> is a partial order $<P,\subseteq>$ with binary lubs $x \cup y$ ($\subseteq$ is called <u>subset</u> and $\cup$ is called <u>binary union</u>). A <u>σ-semilattice</u> is a semilattice with countably infinite lubs $\bigcup x_i$.

<u>Definition 6</u> Let **σSLPos** $(\kappa,\ldots;\lambda,\ldots;\mu,\ldots;\upsilon,\ldots)$ be the category whose objects are structures $<D,\sqsubseteq,\subseteq>$ where $<D,\sqsubseteq>$ is a **Pos** $(\kappa,\ldots;\lambda,\ldots)$ object and $<D,\subseteq>$ is a σ-semilattice such that binary union is μ-continuous and ... (wrt $\sqsubseteq$) and countable union is υ-continuous and ... (wrt $\sqsubseteq$) and whose morphisms are those **Pos** $(\kappa,\ldots;\lambda,\ldots)$ morphisms preserving countable union. In particular set

$$\mathbf{B}_0 = \mathbf{\sigma SLPos}(\omega_0,\omega_1;\omega_0,\omega_1;\omega_0,\omega_1;\omega_0,\omega_1)$$

$$\mathbf{B} = \mathbf{\sigma SLPos}(\omega_0,\omega_1;\omega_0,\omega_1;\omega_0,\omega_1;\omega_1) \qquad \mathbf{B}_1 = \mathbf{\sigma SLPos}(\omega_0,\omega_1;\omega_1;\omega_0,\omega_1;\omega_1)$$

Here $\mathbf{B}_0$ is the nicest category of σ-semilattices we could hope for where even countable union is ω_0-continuous; $\mathbf{B}$ and $\mathbf{B}_1$ are the categories corresponding to $\mathbf{A}$ and $\mathbf{A}_1$ where countable union is ω_1-continuous, but need not be ω_0-continuous (but we do assume binary union ω_0-continuous). Although the morphisms in $\mathbf{B}_1$ are not ω_0-continuous in general, we do have

<u>Lemma 4</u> <u>Quasi-continuity</u> Let f: D $->_1$ E be a $\mathbf{B}_1$-morphism. For any increasing ω_0-chain $a_0 \sqsubseteq a_1 \sqsubseteq \ldots.$ we have $f(\bigsqcup a_n) \subseteq \bigsqcup f(a_n)$.

All these categories are related by various forgetful functors:

$$\begin{tikzcd}
& \mathbf{B}_0 \arrow[dl, "U_0"'] \arrow[d, "W_0"] \\
\mathbf{A} \arrow[d, "V"'] & \mathbf{B} \arrow[l, "U"'] \arrow[d, "W"] \\
\mathbf{A}_1 & \mathbf{B}_1 \arrow[l, "U_1"]
\end{tikzcd}$$

And we expect that the desired powerdomain will be a left-adjoint to U_1 or U or maybe even U_0 or $V \bullet U$.

Theorem 2 The functors U, U_0, V, W have left adjoints, called F, F_0, G, H. However U_1 has no left adjoint.

Proof The positive assertion uses Freyd's Adjoint Functor Theorem [Mac]. The negative one depends on the quasi-continuity lemma. ⊠

So possible powerdomains are $\mathcal{P} \overset{\text{def}}{=} U \bullet F, F_0$ and $\mathcal{P}_1 \overset{\text{def}}{=>} V \bullet U \bullet F \bullet G = V \bullet \mathcal{P} \bullet G$.

Not surprisingly the second is unacceptable because of:

Fact 3 Let D be an $\mathbf{A}$ object and let x and y be two elements of $F_0U_0(D)$. Then if $x \sqsubseteq y$ we have $x \subseteq y$. Further if D has a least element then the converse also holds.

Now we examine the properties of our two candidate powerdomains. In $\mathbf{A}$ we have a morphism $\{\cdot\}$: D -> $\mathcal{P}$D (called *singleton*) which is universal in the sense that to any f: D -> UA there is a unique $\mathbf{B}$-morphism $f^{\dagger}$: FD -> A (the *extension* = *left-adjunct* of f) such that the following diagram commutes.

$$\begin{tikzcd}
D \arrow[d, "\{\cdot\}"'] \arrow[dr, "f"] & \\
\mathcal{P}D \arrow[r, "Uf^{\dagger}"'] & UA
\end{tikzcd}$$

In $\mathbf{A}_1$ analogous remarks hold with $\mathcal{P}_1, \{\cdot\}_1, f^{\dagger 1}, V \bullet U$ and $F \bullet G$ replacing $\mathcal{P}, \{\cdot\}, f^{\dagger}, U$ and F. As an example one can check that $\mathcal{E}(X_\perp) = \mathcal{P}(X_\perp) = \mathcal{P}_1(X_\perp)$. We now try to generalise Fact 1.3.

Definition 7 A $\mathbf{Pos}$-category is a category whose hom-sets are equipped with partial orders so that composition is monotonic. A functor of $\mathbf{Pos}$-categories is *locally-monotonic* (= a $\mathbf{Pos}$-functor) iff it is monotonic on morphisms; it is *locally κ-continuous* if it preserves lubs of κ-chains of morphisms.

Definition 8 Let G: $\mathbf{L}$ -> $\mathbf{K}$ be a $\mathbf{Pos}$-functor. Then f: D -> GA is a *G-orderepi* iff whenever g,g': A -> A' are such that $(Gg) \bullet f \sqsubseteq (Gg') \bullet f$ then $g \sqsubseteq g'$.

Lemma 5 Let G: $\mathbf{L}$ -> $\mathbf{K}$ be a $\mathbf{Pos}$-functor with left-adjoint F such that every f: D -> GA factorises as $D \xrightarrow{f'} GA' \xrightarrow{Gg} GA$ where f' is a G-orderepi. Then the unit ε_D: D -> GFD is a G-orderepi and extension is an isomorphism of partial orders.

Theorem 3 In both $\mathbf{A}$ and $\mathbf{A}_1$ extension is monotonic and preserves lubs of increasing ω_0-and ω_1-chains. Further F and $F \bullet G$ are locally ω_0-and ω_1-continuous $\mathbf{Pos}$ functors. Finally $\mathcal{P}$ and $\mathcal{P}_1$ are locally ω_1-continuous $\mathbf{Pos}$ functors which are not in general ω_0-continuous.

There is no contradiction here with Fact 1.3 as in the first case extension has range in $\underline{\underline{B}}$ and in the second in $\underline{\underline{A}}$. Now we turn to issues involved with the bottom element, $\overline{\perp}$.

Definition 9 $\underline{\underline{A}}^{\perp}$ (respectively $\underline{\underline{A}}_1^{\perp}, \underline{\underline{B}}^{\perp}$) is the full subcategory of $\underline{\underline{A}}$ (respectively $\underline{\underline{A}}_1, \underline{\underline{B}}$) with those objects D containing a least element, $\perp_D$; further $\underline{\underline{A}}_{\perp}$ (respectively $\underline{\underline{A}}_{1\perp}, \underline{\underline{B}}_{\perp}$) is the subcategory of $\underline{\underline{A}}^{\perp}$ (respectively $\underline{\underline{A}}_1^{\perp}, \underline{\underline{B}}^{\perp}$) with the same objects but only those morphisms preserving the least element, the strict ones.

These new categories can be pictured together in terms of a commuting diagram of natural forgetful functors (of which we name six).

$$\begin{tikzcd}
\underline{\underline{A}}_1 & \underline{\underline{A}} \arrow[l, "V"'] & \underline{\underline{B}} \arrow[l, "U"'] \\
\underline{\underline{A}}_1^{\perp} \arrow[u] & \underline{\underline{A}}^{\perp} \arrow[l, "V^{\perp}"] \arrow[u] & \underline{\underline{B}}^{\perp} \arrow[l, "U^{\perp}"] \arrow[u] \\
\underline{\underline{A}}_{1\perp} \arrow[u] & \underline{\underline{A}}_{\perp} \arrow[l, "V_{\perp}"'] \arrow[u] & \underline{\underline{B}}_{\perp} \arrow[l, "U_{\perp}"'] \arrow[u]
\end{tikzcd}$$

The next theorem says that our powerdomain construction also works when these variations are considered.

Theorem 4 If D is an $\underline{\underline{A}}$-object with a least element then FD has a least element too and the singleton function is strict; further extension preserves strictness. Consequently F cuts down to left adjoints $F^{\perp}$ and $F_{\perp}$ of $U^{\perp}$ and $U_{\perp}$ respectively. The corresponding assertions for $\underline{\underline{A}}_1$ also hold.

Solving Domain Equations

To solve recursive domain equations $D \cong F(D)$ one normally proceeds by analogy with fixed-point equations $x = f(x)$ where the solution is given as $\text{Fix}_f = \bigsqcup_{n>0} f^n(\perp)$ and this is justified by the ω_0-continuity of f. What one does is construct the solution as $\text{Fix}_F = \underline{\lim}\ \Delta$ where $\Delta = \langle F^n(\perp), F^n(\perp_{F^n(\perp)})\rangle$ and justify that by the ω_0-continuity of F. Unfortunately neither $\mathcal{P}$ nor $\mathcal{P}$ have the needed continuity property and so we turn to a categorical generalisation of Fact 2, due to Adamek and Koubek [AK]. Below κ is always a limit ordinal.

Definition 10 Let $\underline{\underline{K}}$ be a category. It is a κ-category if it has an initial element, $\perp_{\underline{\underline{K}}}$, and it has direct limits of all $\underline{\lambda}$-chains for $\lambda<\kappa$; for any D we write $\perp_D$ for the unique morphism from $\perp_{\underline{\underline{K}}}$ to D. Let F: $\underline{\underline{K}} \to \underline{\underline{L}}$ be a functor between κ-categories. It is κ-continuous if whenever Δ is an $\underline{\kappa}$-chain and $\rho: \Delta \to D$ is a limiting cone then $F\rho: F\Delta \to FD$ is a limiting cone. Clearly the composition of κ-continuous functors is κ-continuous as are the constant and identity functors.

Definition 11 Let F: $\underline{\underline{K}} \to \underline{\underline{K}}$ be a functor. An F-algebra is a pair $\langle D,\alpha\rangle$ with α: FD -> D. A morphism of F-algebras, f: $\langle D,\alpha\rangle \to \langle E,\beta\rangle$ is any morphism f: D -> E such that the following diagram commutes

$$\begin{tikzcd}
FD \arrow[r, "\alpha"] \arrow[d, "Ff"'] & D \arrow[d, "f"] \\
FE \arrow[r, "\beta"] & E
\end{tikzcd}$$

This clearly gives a category of F-algebras.

Theorem 5 Let $\underline{\underline{K}}$ be a κ-category and suppose F: $\underline{\underline{K}} \to \underline{\underline{K}}$ is κ-continuous. Then the initial F-algebra exists and can be constructed by the following Initial Algebra Construction

1. $D_0 = \perp_{\underline{\underline{K}}}$ and $D_{\lambda+1} = F(D_\lambda)$
2. $f_{01} = \overline{\perp}_{D_1}$ and $f_{\lambda+1,\lambda+2} = F(f_{\lambda,\lambda+1})$
3. For limit λ'', $\langle f_{\lambda,\lambda''}\rangle_{\lambda<\lambda''}: \langle D_{\lambda'}, f_{\lambda,\lambda'}\rangle_{\lambda\leq\lambda'<\lambda''} \to D_{\lambda''}$ is a colimiting cone

4. For limit λ'', $f_{\lambda'',\lambda''+1}$ is the mediating morphism between the universal cone $\langle f_{\lambda+1,\lambda''}\rangle: \langle D_{\lambda+1}, f_{\lambda+1,\lambda'+1}\rangle_{\lambda\leq\lambda'<\lambda''} \to D_{\lambda''}$ and
$\langle Ff_{\lambda,\lambda''}\rangle_{\lambda<\lambda''}: \langle D_{\lambda+1}, f_{\lambda+1,\lambda'+1}\rangle_{\lambda\leq\lambda'<\lambda''} \to F(D_{\lambda''})$

Then $f_{\kappa,\kappa+1}: D_\kappa \to F(D_\kappa)$ is an isomorphism whose inverse gives the initial F-algebra, $\langle D_\kappa, f^{-1}_{\kappa,\kappa+1}\rangle$.

To apply these ideas we generalise [SP] and work in a $\underline{\underline{\text{Pos}}}$-category setting.

Definition 12 Let $\underline{\underline{K}}$ be a $\underline{\underline{\text{Pos}}}$-category. A pair $D \xrightarrow{f} E \xrightarrow{g} D$ is a projection pair (and f is an embedding and g is a projection) if $g \circ f = id_D$ and $f \circ g \sqsubseteq id_E$.

The elementary facts about embeddings and projections are shown in [SP]. In particular every projection g is determined by its corresponding embedding f and we write $g = f^R_E$. Also the embeddings form a category under $\underline{\underline{K}}$-composition which we denote by $\underline{\underline{K}}^E$. Finally both embeddings and projections are strict and so $\underline{\underline{A}}^{\perp E} = \underline{\underline{A}}^E_\perp$ and and $\underline{\underline{A}}_1^{\perp E} = \underline{\underline{A}}^E_{1\perp}$. Note that both have an initial object namely the one-point poset. To see they are both $\underline{\underline{\omega}}_1$-categories one checks both $\underline{\underline{A}}_\perp$ and $\underline{\underline{A}}_{1\perp}$ have both $\underline{\underline{\omega}}_0$-and $\underline{\underline{\omega}}_1$-limits and uses Theorem 6 below.

Definition 13 A $\underline{\underline{\text{Pos}}}(\kappa)$ category is a $\underline{\underline{\text{Pos}}}$-category in which the morphism partial orders have lubs of all increasing κ-chains and where composition is κ-continuous.

Theorem 6 Suppose $\underline{\underline{K}}$ is a $\underline{\underline{\text{Pos}}}(\kappa)$ category with $\underline{\underline{\kappa}}$-limits. Then $\underline{\underline{K}}^E$ has $\underline{\underline{\kappa}}$-colimits and indeed for any $\underline{\underline{\kappa}}$-chain Δ in $\underline{\underline{K}}^E$ we have that $\mu: \Delta \to D$ is colimiting iff $id_D = \bigsqcup_{\lambda<\kappa} \mu_\lambda \bullet \mu^R_\lambda$.

Turning to functors we note that if $T: \underline{\underline{K}}^{op} \times \underline{\underline{L}} \to \underline{\underline{M}}$ is a $\underline{\underline{\text{Pos}}}$-functor then as in [SP] we can define a covariant $T^E: \underline{\underline{K}}^E \times \underline{\underline{L}}^E \to \underline{\underline{M}}^E$ with the same action as T on objects and with $T^E(f,g) \overset{\text{def}}{=\!\!>} T(f^R,g)$ on morphisms.

Corollary 1 Let $\underline{\underline{K}},\underline{\underline{L}},\underline{\underline{M}}$ be $\underline{\underline{\text{Pos}}}(\kappa)$ categories with all limits of $\underline{\underline{\kappa}}$-chains; let $T: \underline{\underline{K}}^E \times \underline{\underline{L}} \to \underline{\underline{M}}$ be a locally κ-continuous functor. Then T^E is κ-continuous.

Now we see that both $\mathcal{P}$ and $\mathcal{P}_1$ and the product and function-space functors all give ω_1-continuous functors. In addition both $\underline{\underline{A}}_\perp$ and $\underline{\underline{A}}_{1\perp}$ have categorical sums which are just the usual smash sums (e.g. see [SP]) these are also locally ω_1-continuous and so give ω_1-continuous functors.

We can therefore follow [HP], say, and obtain a domain of resumptions

$$R \cong S_\perp \to \mathcal{P}(S_\perp + (S_\perp \times R))$$

in $\underline{\underline{A}}$ (to be ranged over by r) and another one (also ranged over by r)

$$R_1 \cong S_\perp \to_1 \mathcal{P}_1(S_\perp + (S_\perp \times R_1))$$

in $\underline{\underline{A}}_1$. Below the isomorphism will be treated as an actual equality for simplicity's sake; similarly we will omit injection functions when dealing with sums. Again we should have used more accurate domain equations to model strictness phenomena, but the extra complications did not seem worthwhile here, and do not affect the theorems in the next section.

4. Denotational Semantics

By using resumptions we attempt to give a denotational semantics to our programming language; the idea will be to model the generative operational semantics. At first we try R; this will fail but R_1 will succeed.

An attempt to use $\underline{\underline{A}}$ To begin we develop a little "categorical programming". Let e_1 be an expression of type $\mathcal{P}(S_\perp + (S_\perp \times R))$ and let e_2 and e_3 be expressions of type $\mathcal{P}(D)$ monotonic in σ and where e_3 is ω_0-and ω_1-continuous in r. Then e = cases e_1 first σ'. e_2 second σ', r'. e_3 is of type $\mathcal{P}(D)$ and abbreviates $[\lambda\sigma' \in S_\perp.\ e_2, \lambda\sigma' \in S_\perp, r' \in R.\ e_3]^\dagger(e_1)$. If e_1,e_2 and e_3 are ω_1-continuous in a variable then so is e (and because of the extension we do not expect ω_0-continuity in general). Again, if e is an expression of type $\mathcal{P}(D)$ monotonic in n (ranging over

$N_\perp$) then the countable choice expression $\bigcup_n e$ abbreviates $(\lambda n \in N_\perp . e)^\dagger N$; this is ω_1-continuous in any variable that e is.

Now we try to define various useful combinators. The definitions are recursive and justified by an appeal to Fact 2.

Flattening The combinator (= operation) $|\cdot| : R \to (S_\perp \to S_\perp)$ is defined recursively by: $|r|\sigma = \underline{\text{cases}}\ r(\sigma)\ \underline{\text{first}}\ \sigma'.\ \{\sigma'\}\ \underline{\text{second}}\ \sigma', r'. |r'|^\dagger(\sigma')$

Composition To model the composition of commands we recursively define a composition combinator ; : $R \to (R \to_1 R)$ by:

$$r_1;r_2(\sigma) = \underline{\text{cases}}\ r_1(\sigma)\ \underline{\text{first}}\ \sigma'.\ \{<\sigma', r_2>\}\ \underline{\text{second}}\ \sigma', r'. \{<\sigma', r';r_2>\}$$

Parallelism We need three combinators corresponding to the three syntactic operators of the generative operational semantics. They are $||_L, ||_R : N_\perp \to R \to_? R \to_? R$ and $||: R \to_? R \to_? R$ where we do not yet know which function spaces are intended. We will see there are no possible choices which make our attempted definitions work.

Try to define $||_L$ and $||_R$ by mutual recursion:

$$\begin{aligned} r_1 ||_L^m r_2(\sigma) = \underline{\text{cases}}\ & r_1(\sigma)\ \underline{\text{first}}\ \sigma'.\{<\sigma', r_2>\} \\ & \underline{\text{second}}\ \sigma', r'.\ \underline{\text{if}}\ m{=}0\ \underline{\text{then}} \bigcup_n \{<\sigma', r' ||_R^n r_2>\} \\ & \quad \underline{\text{else}}\ \{<\sigma', r' ||_L^{m-1} r_2>\} \end{aligned}$$

($||_R$ is defined symmetrically).

If these definitions were legitimate we would then go on to define the parallel combinator by

$$r_1 || r_2(\sigma) = \bigcup_n r_1 ||_L^n r_2(\sigma) \cup \bigcup_n r_1 ||_R^n r_2(\sigma)$$

However the definitions cannot be acceptable. For example in the definition of $||_L$ in order that the conditional expression be ω_0-continuous in r' it is necessary that $r_1 ||_R^m r_2$ be ω_0-continuous in r_1; but r_1 occurs in both the "first" and "second" branches of the definition of $||_R$ and so such continuity cannot be guaranteed. Despite some effort it was not found possible to produce any acceptable definitions and for that reason the attempt to use $\underline{\underline{A}}$ seems doomed to failure.

Using $\underline{\underline{A}}_1$ Here one tries the domain R_1. The cases construction $\underline{\text{cases}}\ e_1\ \underline{\text{first}}\ \sigma'.\ e_2\ \underline{\text{second}}\ \sigma', r'.\ e_3$ is introduced as above but now only ω_1-continuity of e_3 in r' is required; it abbreviates $[\lambda\sigma' \in S_\perp . e_2, \lambda\sigma' \in S_\perp, r' \in R_1 . e_3]^{\dagger 1}(e_1)$. The countable union construction $\bigcup_n e$ is introduced as above and abbreviates $(\lambda n \in N_\perp\ e)^{\dagger 1}(N)$.

The flattening combinator $|\cdot| : R_1 \to_1 (S_\perp \to_1 S_\perp)$ and the composition combinator ; : $R_1 \to_1 R_1 \to_1 R_1$ are defined analogously to before and now the analogous definitions for the parallel combinators $||_L, ||_R : N_\perp \to_1 R_1 \to_1 R_1 \to_1 R$ and $||: R_1 \to_1 R_1 \to_1 R_1$ are legitimate.

We are at last in a position to give the denotational semantics of our programming language. The denotational function $\mathcal{C}$: gCom $\to R_1$ is defined by structural induction on commands:

$$\begin{aligned}
\mathcal{C}[\![\text{ac}]\!] &= \lambda\sigma \in S_\perp .\ \{\mathcal{A}[\![\text{ac}]\!](\sigma)\}_1 \\
\mathcal{C}[\![\underline{\text{skip}}]\!] &= id_{S_\perp} \\
\mathcal{C}[\![c_1;c_2]\!] &= \mathcal{C}[\![c_1]\!];\ \mathcal{C}[\![c_1]\!] \\
\mathcal{C}[\![\underline{\text{if}}\ b\ \underline{\text{then}}\ c_1\ \underline{\text{else}}\ c_2]\!] &= \lambda\sigma \in S_\perp .\ \underline{\text{if}}\ \mathcal{B}[\![b]\!](\sigma)\ \underline{\text{then}}\ \{<\sigma, \mathcal{C}[\![c_1]\!]>\}_1 \\
&\qquad \underline{\text{else}}\ \{<\sigma, \mathcal{C}[\![c_2]\!]>\}_1 \\
\mathcal{C}[\![\underline{\text{while}}\ b\ \underline{\text{do}}\ c]\!] &= \mu r \in R_1 . \lambda\sigma \in S_\perp .\ \underline{\text{if}}\ \mathcal{B}[\![b]\!](\sigma) \\
&\qquad \underline{\text{then}}\ \{<\sigma, \mathcal{C}[\![c]\!];m>\}_1\ \underline{\text{else}}\ \{\sigma\}_1 \\
\mathcal{C}[\![c_1 ||^m c_2]\!] &= \mathcal{C}[\![c_1]\!] ||_L^m \mathcal{C}[\![c_2]\!]
\end{aligned}$$

$$\mathcal{C}[\![c_1 \,||_m\, c_2]\!] = \mathcal{C}[\![c_1]\!] \,||_R^m\, \mathcal{C}[\![c_2]\!]$$
$$\mathcal{C}[\![c_1 \,||\, c_2]\!] = \mathcal{C}[\![c_1]\!] \,||\, \mathcal{C}[\![c_2]\!]$$

Here if e is an expression of type D that is ω_1-continuous in a variable x of type D then $\mu x \in D.e$ is the least x=e; it is ω_1-continuous in any variable that e is.

Relation with the operational semantics

The resumption semantics was introduced as an abstract version of the operational semantics. To formalise this we define Op; gCom -> R_1 by

$$\mathrm{Op}[\![c]\!](\sigma) = \bigcup\{\{<\sigma', \mathrm{Op}[\![c']\!]>\}_1 | \exists a <c,\sigma> \xrightarrow{a}_g <c',\sigma'>\} \cup \bigcup\{\{\sigma'\}_1 | \exists a <c,\sigma> \xrightarrow{a}_g \sigma'\}$$

This definition is easily justified. Now we see that the operational semantics determines the denotational semantics.

Theorem 7 $\mathcal{C}$ = Op.

The proof of this theorem makes heavy use of a form of Scott Induction which we call ω_1-induction (and contrast that with the usual ω_0-induction). A property $P \subseteq D$ is ω_0-(ω_1-) inductive if it has lubs of increasing ω_0-(respectively ω_1-) chains. The ω_1-induction rule is:

$$\frac{P(\bot) \quad \forall x\, P(x) \supset P(e)}{P(\mu x.e)}$$

provided P is both ω_0-and ω_1-inductive and $\lambda x.e$ is ω_1-continuous. What we hope is that ω_1-induction will prove as useful a tool for handling countable non-determinism as ω_0-induction has proved for sequential programming.

Finally we see that the operational semantics of section 2 can be obtained from the denotational semantics.

Theorem 8 1. For any c in Com and states σ, σ', $\sigma R[\![c]\!] \sigma'$ iff $\sigma' \in |\mathcal{C}[\![c]\!]|(\sigma)$

2. For any c in Com and state σ, $\sigma \in T[\![c]\!]$ iff $\bot \notin |\mathcal{C}[\![c]\!]|(\sigma)$

Acknowledgements

I would like to thank Axel Poigné, Matthew Hennessy and Robin Milner for useful discussions, and Eleanor Kerse for the typing. The work was supported with the aid of an SERC grant.

References

[AK] Adamek, J. and Koubek, V. Least fixed points of a functor. JCSS, Vol. 19, No. 2, pp. 163-178, (1979).

[AO] Apt, K.R. and Olderog, E.-R. Proof rules dealing with fairness. Bericht Nr. 8104, Institut für Informatik und Praktische Mathematik, Kat. Christian-Albrechts Universität, (1981).

[AP] Apt, K.R. and Plotkin, G.D. A Cook's tour of countable non-determinism. Proc. ICALP 1981. LNCS Vol. 115 (eds. S. Even and O. Kariv). Berlin: Springer-Verlag, pp. 479-494, (1981).

[Bar] Barendregt, H.P. The lambda calculus, its syntax and semantics. Studies in Logic, Vol. 103, (1981). Amsterdam: North-Holland.

[HP] Hennessy, M.C.B. and Plotkin, G.D. Full abstraction for a simple parallel programming language. Proc. MFCS, LNCS Vol. 74, pp. 108-120 (ed. Becvar,J.) (1979), Berlin: Springer-Verlag.

[Kel] Keller, R. A fundamental theorem of asynchronous parallel computation in parallel processing, LNCS Vol. 24 (ed. T. Feng) Berlin: Springer-Verlag.

[Kwo] Kwong, Y.S. On the absence of livelock in parallel programs. Semantics of concurrent computation, LNCS Vol. 70, pp. 172-190 (ed. G. Kahn) Berlin: Springer-Verlag, (1979).

[LPS] Lehmann, D., Pnueli, A. and Stavi, J. Impartiality, justice and fairness: the ethics of concurrent termination. Proc. ICALP 1981, LNCS Vol. 115 (eds. S. Even and O. Kariv) Berlin: Springer-Verlag, pp. 264-277, (1981).

[Mac] MacLane, S. Categories for the Working Mathematician. Berlin: Springer-Verlag, (1971).

[Man] Manna, Z. Logics of programs. Proc. IFIP Congress 1980.

[Mil] Milner, R. A calculus of communicating systems. LNCS Vol. 92,(1980) Berlin: Springer-Verlag.

[Par] Park, D. A predicate transformer for weak fair iteration. Proc. 6th IBM Symposium on Mathematical Foundations of Computer Science, Hakone, Japan, (1981).

[Plo1] Plotkin, G.D. A powerdomain construction. SIAM Journal on Computation, Vol. 5, No. 3, pp. 452-487, (1976).

[Plo2] Plotkin, G.D. A structural approach to operational semantics. DAIMI FN-19. Computer Science Department, Aarhus University, (1981).

[Smy] Smyth, M.B. Powerdomains. JCSS, Vol. 16, No. 1, (1978).

[SP] Smyth, M. and Plotkin, G.D. The categorical solution of recursive domain equations. SIAM Journal on Computation. To appear. (1981).