

A Domain-theoretic Model for a Higher-order Process Calculus

Radhakrishnan Jagadeesan and Prakash Panangaden*
Computer Science Department, Cornell University

March 21, 1996

Abstract

In this paper we study a *higher-order* process calculus, a restriction of one due to Boudol, and develop an abstract, model for it. By abstract we mean that the model is constructed domain-theoretically and reflects a certain conceptual viewpoint about observability. It is not constructed from the syntax of the calculus or from computation sequences. We describe a new powerdomain construction that can be given additional algebraic structure that allows one to model concurrent composition, in the same sense that Plotkin's powerdomain can have a continuous binary operation defined on it to model choice. We show that the model constructed this way is adequate with respect to the operational semantics. The model that we develop and our analysis of it is closely related to the work of Abramsky and Ong on the lazy lambda calculus.

1 Introduction

A fundamental problem in the semantics of parallel programming languages is integrating concurrency with abstraction. Kahn's pioneering work on static dataflow [10] is an example where concurrency meshes smoothly with abstraction. More precisely, in Kahn's model one can abstract away the internal, operational details of processes and view them as continuous stream-functions that compose as functions should. Feedback is modeled by fixed-point iteration. This is a very pleasant application of Scott's semantic ideas.

In almost any elaboration of Kahn's model, the situation becomes much more difficult. In the context of indeterminate dataflow, recent work by Kok, Jonsson and others has shown that one gets fully abstract models from the traces of computations [9, 11, 17]. Traces do not, however, give one the same level of abstraction that is provided by being able to think of processes as functions. Similarly, though process algebra has now reached a high degree of mathematical maturity and elegance, see, for example, the recent books by Milner and by Hennessy [7, 12], it remains essentially an operational analysis of processes. The semantic models available are constructed from the computations of terms.

*This research supported by NSF grant CCR-8818979

Our study of the restricted version of Boudol’s calculus, henceforth called the γ -calculus, is based on viewing the communication ability of processes as the fundamental observable. A process that is diverging has no communication ability, a process that can accept a single input and then diverges has more communication ability. This is, in some sense, a natural extension of the idea of making convergence the basic observable in the λ -calculus. The connection with the lazy λ -calculus comes about by observing that the presence of an outer λ -abstraction signifies that a term has communication ability. Clearly, we should distinguish $\lambda x.\Omega$ from Ω , where Ω represents any divergent term such as $(\lambda x.xx)(\lambda x.xx)$, since they have different communication abilities. Thus we need our model to resemble the models of the lazy λ -calculus [1] rather than the models discussed by Scott and Wadsworth [20].

We model concurrency in the γ -calculus as indeterminate interleaving. Thus, we need to deal with the fact that a term may or may not converge. We need two predicates to capture the convergence properties of term; these are “may converge”, written \Downarrow^{may} , and “must converge”, written \Downarrow^{must} . The operational preorder is defined in terms of these predicates.

The main contribution in this paper is the new powerdomain construction that we describe. Roughly speaking, it allows us to model processes as sets of functions and allows us to capture the notions of observability described above. Intuitively, the key difference between our powerdomain and the Plotkin powerdomain [15] is that our construction is defined to work on functions spaces. One cannot, of course, expect the last remark to be taken literally since when one is handed a domain, even one that is a function space, it may not be presented as a function space. The recursive domain equation that we solve uses a functor that first builds a function space and then carries out certain constructions on the result. This functor is *formally* defined on all of NSFP but the constructions are clearly motivated by viewing the finite elements as finite sets of functions.

One important point worth stressing early is why we defined a new powerdomain instead of using the Plotkin powerdomain. Both powerdomains would yield an adequate model and, as far as we know, neither yields a fully abstract model. Nevertheless, we feel that our model comes closer to capturing the operational properties of the calculus. When one attempts to use the Plotkin powerdomain to model the γ calculus, one finds that certain operational laws are violated in the model. One can define an operational preorder on terms that embodies the above notions of observability. It turns out that this preorder meshes well with the partial order in our model in the following sense. Suppose that we define a preorder on domain elements that formally imitates the definition of the operational preorder. Then, we recover exactly the original partial order in the model. This does not happen with the Plotkin powerdomain. More concretely, we exhibit terms that are deemed equal by the operational semantics, such that the meanings of these terms are unrelated in a model based on the Plotkin powerdomain, and are the same in our semantics. Thus, our model, though probably not fully abstract, describes the interplay between choice, lambda abstraction and concurrency in a smooth way. This discussion is made precise in sections 3 and 4.

The paper is organised as follows. We introduce the γ -calculus and discuss it informally. In section 2 we define a sub-calculus, its operational semantics and introduce the operational preorder. In section 3 we give the powerdomain construction. In section 4 we delineate the algebraic properties of the model. In section 5 we describe the adequacy properties of the model. In the final section we mention related work and directions for further study.

The γ Calculus

In this section we quickly review Boudol's γ -calculus and describe an example of a simple concurrent program expressed in it. The key contribution of this calculus is to provide a smooth integration of concurrent communication concepts with functional abstraction. Boudol's original work [5] describes the calculus and shows how the lazy λ -calculus is embedded in it.

Let C be a set of channel names. Terms are generated by the grammar:

$$Terms ::= x \mid (\lambda_1 x_1 \mid \dots \mid \lambda_k x_k).p \mid$$

$$p \odot q \mid p|q \mid \bar{\lambda}p.1 \mid 1$$

where $\lambda, \lambda_1 \dots \lambda_k$ are (not necessarily distinct) members of C . The novel constructs here are $(\lambda_1 x_1 \mid \dots \mid \lambda_k x_k)$, \odot , $\bar{\lambda}$ and $p|q$. The term 1 represents the terminated process. It is the identity for both \odot and $|$. Roughly speaking, $(\lambda_1 x_1 \mid \dots \mid \lambda_k x_k).p$ means that p waits *concurrently* for k unordered values. The combinator $|$ represents concurrency. The intuitive meaning of $p|q$ is that p and q are juxtaposed, *without any communication between them*. The term $\bar{\lambda}p.1$ represents a process that outputs p on channel λ and terminates. Finally, $p \odot q$ means that p and q communicate on *all* channels. The processes p and q cannot communicate with any other process until one of them terminates. The key points to note are that $|$ represents pure concurrency without any interaction while \odot represents a very tight interaction between processes. The \odot serves as a generalization of application. The communication is effected in the manner now customary in process algebras, one matches a name with its *dual* name. Finally there is no construct like $\bar{\lambda}p.M$ where M is a term that is not 1. An output term cannot produce a value and go on to do something else.

The following simple term that appears in Boudol's original paper [5], illustrates some of the features of the γ -calculus.

$$A \simeq \lambda y.\alpha x.(\bar{\beta}z|(y \odot \bar{\lambda}y))$$

Now consider $A \odot \bar{\lambda}A$. This term reduces in one step to

$$\alpha x.(\bar{\beta}z|(A \odot \bar{\lambda}A))$$

This last term has the property that it waits for a signal on α then outputs z on β and reproduces itself. It is a term that repeatedly offers communication to the outside.

2 Operational Semantics

In this section we define the restricted calculus. From the point of view of difficulty of modeling we have eliminated the possibility of deadlock but we still have indeterminacy and concurrency. We do not allow \odot in its unrestricted form. We force it to look like applications. More precisely, the \odot construct can only be used in the combination $\lambda x.M \odot \bar{\lambda}P$. Thus it cannot be introduced in a case where there is no communication possibility as in $\lambda x.x \odot \lambda x.x$.

The terms are generated by the grammar

$$Terms ::= x \mid \lambda(x_1 \dots x_k).p \mid (pq) \mid p|q$$

We do not use the \odot symbol explicitly, it is implicitly present in the applications (pq) . We usually drop the parenthesis from (pq) . We use Λ_0 for the terms that do not have free variables, and call members of this set closed terms.

Definition 1. The syntactic equality \equiv is the congruence generated by the equation:

$$p|(q|r) \equiv (p|q)|r$$

The reduction rules are as follows.

- $(M_1 | \dots | \lambda \langle x_1 \dots x_k \rangle . M | \dots | M_n) N \longrightarrow M_1 | \dots | \lambda \langle x_1 \dots x_{i-1}, x_{i+1} \dots x_k \rangle . [x_i \mapsto N] M | \dots | M_n$, if $1 \leq i \leq k$
- $M \longrightarrow M' \Rightarrow M|N \longrightarrow M'|N$
- $N \longrightarrow N' \Rightarrow M|N \longrightarrow M|N'$
- $M \longrightarrow M' \Rightarrow MN \longrightarrow M'N$

Recall that the intuitive meaning that was assigned to $\lambda x.M$ was the presence of a communication ability on port λ . We take the point of view that the only observable behaviour about a process is the acceptance of values on channels. So, we attempt to set up a theory that “measures” the communication ability of a term. The study of the lazy lambda calculus [1, 14], proceeds on very similar lines. There the “definedness” of a term is measured by its outermost abstractions or, in other words, how many arguments it can accept. This is exactly what we do except that we need to confront the indeterminacy in the reduction relation. The study of the lazy λ -calculus motivates the definition of a convergence predicate. Notice that the presence of non-determinism means that for a given term M , we might have both the following situations:

- $M \xrightarrow{*} \lambda \langle x_1 \dots x_k \rangle . M'$
- An infinite reduction sequence
 $M = M_0 \longrightarrow M_1 \longrightarrow M_2 \dots$

We define predicates \Downarrow^{may} read as “may converge” and \Downarrow^{must} read as “must converge”.

$M \Downarrow^{may}$ is intended to indicate that M can accept input on channel λ after a (possibly empty) finite sequence of silent actions. This can be viewed as a kind of “partial correctness assertion” about M .

Definition 2. Define a set S of terms inductively as follows:

1. $\lambda \langle x_1 \dots x_k \rangle . M \in S$, $1 \leq k$, $(\forall \lambda \langle x_1 \dots x_k \rangle . M \in \Lambda_0)$
2. $[M \in S] \Rightarrow (\forall N \in \Lambda_0) [M|N \in S]$

$M \Downarrow^{may}$ if $(\exists M') [M \xrightarrow{*} M' \wedge M' \in S]$

To model total correctness guarantees on terms, we need to be able to say that a term M “always accepts input on channel λ ”, as opposed to the M “can accept input on channel λ ” assertion that motivated \Downarrow^{may} . In the setting of the subcalculus with only one channel, this is equivalent to saying that M has no infinite silent computation.

Definition 3. $M \Downarrow^{must}$ if there is no infinite reduction sequence $M = M_0 \longrightarrow M_1 \longrightarrow M_2 \dots$

The predicates \Downarrow^{may} and \Downarrow^{must} are the primitive observables in the calculus. Two terms that “behave” similarly in all contexts with respect to this notion of observation, are not to be differentiated. So, we define an operational preorder \preceq_c , in the style of definitions of contextual precongruence in the setting of the lambda calculus [2].

Definition 4. The relation \preceq_c on Λ_0 is defined by $M \preceq_c N \Leftrightarrow (\forall C[\cdot] \in \Lambda_0)$, the following hold:

1. $C[M] \Downarrow^{may} \Rightarrow C[N] \Downarrow^{may}$
2. $C[M] \Downarrow^{must} \Rightarrow C[N] \Downarrow^{must}$

The idea of \preceq_c is extended to open terms in the usual way. Let M, N be terms such that the free variables of M and N are contained in $\{x_1 \dots x_n\}$. Then, $M \preceq_c N$ if for all possible substitutions $P_1 \dots P_n$ of closed terms for $\{x_1 \dots x_n\}$, we have $[x_1 \mapsto P_1 \dots x_n \mapsto P_n] M \preceq_c [x_1 \mapsto P_1 \dots x_n \mapsto P_n] N$.

We now define a preorder \preceq_b that relates the communication abilities of terms in purely applicative contexts. It turns out that the preorders \preceq_b and \preceq_c coincide. This simplifies operational proofs of equality of terms.

Define (on closed terms):

1. $M \preceq_0 N$ if
 - $M \Downarrow^{may} \Rightarrow N \Downarrow^{may}$
 - $M \Downarrow^{must} \Rightarrow N \Downarrow^{must}$
2. $M \preceq_{k+1} N$ if
 - $M \preceq_k N$
 - $(\forall P) [MP \preceq_k NP]$

Definition 5. $\preceq_b = \bigcap \preceq_k, k \in \omega$

The relation \preceq_b has an alternate characterisation as the greatest fixed point of a monotone functional.

Lemma 1. Define a function F on binary relations of closed terms by:
 $M F(R) N$ if

- $M \preceq_0 N$

- $(\forall P \in \Lambda_0) [(MP, NP) \in R]$

Then, \preceq_b is the maximum fixed point of F

The key point to note is that the relations \preceq_b and \preceq_c coincide.

Theorem 1. (Operational extensionality) $M \preceq_b N \Leftrightarrow (\forall C[.]) [C[M] \preceq_b C[N]]$

To prove this theorem, we first prove that the operator $|$ is monotone with respect to \preceq_b . This proof involves a detailed analysis of reductions and is given in the full paper [8]. Theorem 1 is now proved by a variant of the proofs of operational extensionality in lambda calculi [3].

A small example helps to illustrate the nature of the preorder \preceq_b . We abbreviate the term $\lambda\langle x, y \rangle.x$ as *or* and write it in infix form for readability. Let M denote the term $\lambda x.[\Omega \text{ or } \lambda y.\Omega]$. Let N denote the term $[\lambda xy.\Omega \text{ or } \lambda x.\Omega]$. Then, a simple proof on the inductive definition of \preceq_b shows that M and N are equivalent under the operational preorder. This identification of choices made “before” and “after” an abstraction will play a key role in the development of our domain theoretic semantics.

3 The Powerdomain Construction

In this section we define the powerdomain construction that we use. We introduce it as a functor in a certain category of nondeterministic continuous algebras. We obtain a model of the γ -calculus by constructing a solution to a recursive domain equation in the usual way [18]. All proofs are omitted. A complete account is contained in the full paper [8].

Many of the ideas are the same as in the analysis of the lazy λ -calculus but the details are somewhat more complicated. Before we begin with the mathematical details we discuss some motivational issues. As the adequacy proof shows, semantic equality in our model is at least as fine as the equality induced by the operational preorder. We are almost certain that one could construct an adequate model for the fragment of the γ -calculus that we consider using the Plotkin powerdomain [15]. Why, then, did we choose to use this powerdomain rather than Plotkin’s?

Our model is probably not fully abstract but it is, in some sense, “closer to being fully abstract” than a model based on the Plotkin powerdomain would be. In order to clarify this point, consider the example discussed at the end of the previous section. The terms $\lambda x.[\Omega \text{ or } \lambda y.\Omega]$ and $\lambda xy.\Omega \text{ or } \lambda x.\Omega$ were deemed equivalent by the operational semantics. In order to avoid confusion we use the notation $up(x \mapsto e)$ to represent lifted functions in the semantic domains. Intuitively, we expect the term Ω to denote – in the semantic model. Thus the denotations of the pair of terms above are $\{up(x \mapsto \{-, up(y \mapsto -)\})\}$ and $\{up(x \mapsto -), up(y \mapsto up(x \mapsto -))\}$. It is easy to check that these are not Egli-Milner related. Thus, a model based on the Plotkin powerdomain would not identify these terms. The difference arises from the way we order the finite sets. We do not use the Egli-Milner order, rather we use the fact that we have sets of functions and use an order that reflects the applicative behaviour of the sets.

The intuition is that the partial order of the domain must satisfy the defining equations of the operational preorder. This idea can be treated formally by introducing generalised versions of quasi-applicative transition systems used in the study of the lazy lambda calculus [1, 14]. We restrict ourselves to an informal discussion. Define semantic versions of the convergence predicates as follows:

- $f \downarrow^{\mathbf{may}}$ if $f \neq -$
- $f \downarrow^{\mathbf{must}}$ if $- \notin f$

From lemma 1, the operational preorder \preceq_b satisfies :

$$M \preceq_b N \Leftrightarrow \begin{cases} M \downarrow^{\mathbf{must}} \Rightarrow N \downarrow^{\mathbf{must}} \\ M \downarrow^{\mathbf{may}} \Rightarrow N \downarrow^{\mathbf{may}} \\ (\forall P) MP \preceq_b NP \end{cases}$$

Let \diamond be notation for the semantic application function. Thus we would like the partial order of the domain to satisfy:

$$f \sqsubseteq g \Leftrightarrow \begin{cases} f \downarrow^{\mathbf{may}} \Rightarrow g \downarrow^{\mathbf{may}} \\ f \downarrow^{\mathbf{must}} \Rightarrow g \downarrow^{\mathbf{must}} \\ \forall x. f \diamond x \sqsubseteq g \diamond x \end{cases}$$

The above holds in our model but not in the model based on the Plotkin powerdomain. In the Plotkin powerdomain, only the left to right implication holds. This point can be clarified further. Define a preorder \leq on the elements of the semantic domain as follows. The definition mimicks the definition 5 of the operational preorder \preceq_b

- $f \leq_0 g$ if
 - $f \downarrow^{\mathbf{may}} \Rightarrow g \downarrow^{\mathbf{may}}$
 - $f \downarrow^{\mathbf{must}} \Rightarrow g \downarrow^{\mathbf{must}}$
- $f \leq_{k+1} g$ if
 - $f \leq_k g$
 - $(\forall x) [f \diamond x \leq_k g \diamond x]$
- $\leq = \bigcap \leq_k, k \in \omega$

In our model, \leq coincides with the partial ordering of the domain \sqsubseteq . In the model based on Plotkin powerdomain, the domain ordering \sqsubseteq is a strict refinement of the ordering \leq .

Basic notation

All domains in this section are SFP objects. We use $B(D)$ as notation for the basis of D . We follow the notation of [1, 14]. Let D, E objects of CPO . Let $f \in D \rightarrow E$.

- D_- is the cpo defined as follows:

- $|D_-| = \{-\} \cup \{\langle 0, d \rangle \mid d \in D\}$
- Let $y, z \in D_-$. Then
 - $y \sqsubseteq z \Leftrightarrow y = - \vee [y = \langle 0, d_1 \rangle \wedge z = \langle 0, d_2 \rangle \wedge d_1 \sqsubseteq_D d_2]$
- If $d \in D$, define $up(d) = \langle 0, d \rangle$
- Define $lift(f) \in D_- \rightarrow E$ by:
 - $lift(f)(-) = -_E$
 - $lift(f)(\langle 0, d \rangle) = f(d)$
- Let $dn_D = lift(id_D)$
- $\diamond : (D_1 \rightarrow D_2)_- \times D_1 \rightarrow D_2$, the application operation is defined by
 - $- \diamond x = -$
 - $up(f) \diamond x = f(x)$, where $f \in D_1 \rightarrow D_2$

Definition 6. $\langle D, \star \rangle$ is a continuous algebra if \star is a continuous function in $D \times D \rightarrow D$, satisfying upper semi-lattice axioms. (idempotence, commutativity, associativity)

Definition 7. Let $\langle D_1, \star_1 \rangle$ and $\langle D_2, \star_2 \rangle$ be continuous algebras. Let $f \in D_1 \rightarrow D_2$. f is said to be **linear** if
 $(\forall \{x_1, x_2\} \subseteq D_1) [f(x_1 \star_1 x_2) = f(x_1) \star_2 f(x_2)]$

Definition 8. Let $\langle D_1, \star_1 \rangle$ and $\langle D_2, \star_2 \rangle$ be continuous algebras. Then, (e, p) is a linear embedding-projection pair if the following hold:

- $p \circ e = 1_{D_1}$
- $e \circ p \sqsubseteq 1_{D_2}$
- e is linear
- p is linear

Given any poset D , If $s = \{f_1 \dots f_n\}$ where $(\forall i) [1 \leq i \leq n] [f_i \in (D \rightarrow D)_-]$, and $x \in D$ then $s \diamond x$ is notation for $(f_1 \diamond x) \star (f_2 \diamond x) \dots \star (f_n \diamond x)$.

3.1 The Powerdomain Functor

In this subsection we define the powerdomain functor and show that it is continuous on a category of algebras very closely related to the bifinites (SFP).

Definition 9. Let $\langle D, \star \rangle$ be a continuous algebra. Then $P(D)$ is a preorder defined as follows:

- $|P(D)| = \{s \mid s \in P_{fin}(B((D \rightarrow D)_-))\}$

- $s_1 \sqsubseteq s_2 \Leftrightarrow$
 1. $- \in s_2 \Rightarrow - \in s_1$
 2. $s_1 \neq \{-\} \Rightarrow s_2 \neq \{-\}$
 3. $(\forall x \in D)[s_1 \diamond x \sqsubseteq s_2 \diamond x]$

$\overline{P}(D)$ is the ideal completion of $P(D)$. We can define a union operation on $\overline{P}(D)$ to make it a continuous algebra. Define \uplus from $P(D) \times P(D)$ to $P(D)$ by $s_1 \uplus s_2 = s_1 \cup s_2$. \uplus is monotone in each argument. and can be extended to a continuous function from $\overline{P}(D) \times \overline{P}(D)$ to $\overline{P}(D)$. Furthermore, \uplus satisfies upper semi-lattice axioms.

We hope to solve the recursive domain equation $D \simeq \overline{P}(D)$. So, we need to establish a suitable category in which the above construction generalises to a functor preserving colimits of ω -chains. Define the category *NSFP* as follows:

- Objects:
The objects are continuous algebras expressible as the colimits of ω -chains of finite continuous algebras, where the arrows of the chain are linear embedding-projection pairs.
- Arrows:
The arrows are linear embedding projection pairs.

In particular, all the objects are *SFP* objects. The above category can be viewed intuitively as that obtained by adding colimits of countable directed diagrams of finite continuous algebras, where the arrows of the diagram are linear embedding-projection pairs. Also note that the category is a subcategory of *SFP^{ep}* that contains the image of the Plotkin-powerdomain functor acting on *SFP^{ep}*, where *SFP^{ep}* is the category of SFP objects with arrows embedding-projection pairs. The following lemma is easy to prove.

Lemma 2. • *NSFP* is closed under colimits of countable directed diagrams

- The one element domain is the initial object

The recursive domain equation $D \simeq \overline{P}(D)$ is solvable in *NSFP* if we can prove that $\overline{P}(\cdot)$ is a functor on *NSFP* that preserves colimits of ω -chains. We now define the action of $\overline{P}(\cdot)$ on linear embedding projection pairs. Let $\langle D_1, \star_1 \rangle$ and $\langle D_2, \star_2 \rangle$ be continuous algebras. Let (e, p) be a linear embedding-projection pair. Define e' as follows:

- Define $e' : P(D_1) \rightarrow \overline{P}(D_2)$ by:
 - $e'(-) = -$
 - $e'(\langle 0, f \rangle) = up(e \circ dn(f) \circ p)$
 - $e'\{F_1 \dots F_n\} = \{e'(F_1) \dots e'(F_n)\}$

e' is well-defined and monotone and extends uniquely to a continuous function from $\overline{P}(D_1)$ to $\overline{P}(D_2)$. Furthermore, it is easy to check that e' is linear. p' is defined from p similarly. It is not too hard to show that $\langle e', p' \rangle$ are a linear embedding-projection pair.

Now we have the machinery to define the arrow part of the functor. Define $\overline{P}((e, p)) = (e', p')$. It is easy to check that

- $\overline{P}((id_D, id_D)) = (id_D, id_D)$, for any continuous algebra $\langle D, \star \rangle$
- Let (e_1, p_1) be a linear embedding projection pair between $\langle D_1, \star_1 \rangle$ and $\langle D_2, \star_2 \rangle$. Let (e_2, p_2) be a linear embedding projection pair between $\langle D_2, \star_2 \rangle$ and $\langle D_3, \star_3 \rangle$. Then $(e_1 \circ e_2, p_2 \circ p_1)$ is a linear embedding projection pair between $\langle D_1, \star_1 \rangle$ and $\langle D_3, \star_3 \rangle$ and we have $\overline{P}(e_2 \circ e_1, p_1 \circ p_2) = \overline{P}((e_2, p_2)) \circ \overline{P}((e_1, p_1))$

The final lemma establishes that this functor is continuous and thus one can solve recursive domain equations using it.

Lemma 3. Let $\Delta = \langle D_m, \langle f_{mn}, f_{nm} \rangle \rangle$ be a chain of linear embedding projection pairs. Let $\langle D, \rho \rangle = \text{Colim } \Delta$. Then, $\text{Colim } \overline{P}(\Delta) \simeq \overline{P}(\text{Colim } \Delta)$, where $\text{Colim } \overline{P}(\Delta)$ means $\text{Colim } \langle \overline{P}(D_m), \overline{P}(\langle f_{mn}, f_{nm} \rangle) \rangle$.

4 The Model and its Basic Properties

In this section we define the model and state some basic properties. The model of the subset of the gamma calculus that we are considering, is the initial solution to the recursive domain equation

$$D \simeq \overline{P}(D).$$

The initial solution is constructed in the usual way. A standard index calculation analysis [14, 2] proves that the initial solution meets the conditions that motivated the construction. Note that the following condition is analogous to the property termed ‘‘Conditional strong extensionality’’ in the setting of the lazy lambda-calculus [1, 14].

Lemma 4. (Conditional Strong extensionality)

Let $d, e \in D$. Then $d \sqsubseteq e \Leftrightarrow$

1. $d \neq - \Rightarrow e \neq -$
2. $- \notin d \Rightarrow - \notin e$
3. $(\forall x \in D) [d \diamond x \sqsubseteq e \diamond x]$

The $|$ constructor is modelled by a continuous function $\times : D \times D \rightarrow D$. Let D_s be the iterates in the solution of the recursive domain equation $D \simeq \overline{P}(D)$. Define a family of functions $\times_{(s,t)} : D_s \times D_t \rightarrow D_{(s+t)}$, by induction on $s + t$ as follows. Let $f \in D_s$, $g \in D_t$. Let f_i denote the projections of f on the i th iterate D_i .

- $(s + t = 0)$. $f \times_{(0,0)} g = -_0$
- $(s + t \neq 0)$. Assume f, g are singleton sets. Then, define by cases on $[f]_1, [g]_1$ as follows.
 - $\{\lambda x. -_0\} = [f]_1, -_0 = [g]_1$. Then,

$$f \times_{(s,t)} g = -_{(s+t)} \star \text{up}[x \in D_{s+t-1} \mapsto (f \diamond x) \times_{(s-1,t)} -_t]$$
 - $\{\lambda x. -_0\} = [g]_1, -_0 = [f]_1$. Then,

$$f \times_{(s,t)} g = -_{(s+t)} \star [\text{up}[x \in D_{s+t-1} \mapsto -_s \times_{(s,t-1)} (g \diamond x)]]$$
 - $\{\lambda x. -_0\} = [f]_1, \{\lambda x. -_0\} = [g]_1$. Then,

$$f \times_{(s,t)} g = \text{up}[x \in D_{s+t-1} \mapsto ((f \times_{(s,t-1)} (g \diamond x)) \star [(f \diamond x) \times_{(s-1,t)} g])]$$
 - $-_0 = [f]_1, -_0 = [g]_1$. Then,

$$f \times_{(s,t)} g = -_{(s+t)}.$$
- $(s + t \neq 0)$. $f = \{f_1 \dots f_m\}, g = \{g_1 \dots g_n\}$. Then,

$$f \times_{(s,t)} g = \star [f_i \times_{(s,t)} g_j | 1 \leq i \leq m, 1 \leq j \leq n]$$

We can show that $\times_{(s,t)}$ is well-defined and monotone in both its arguments. It can also be shown that the subscripts can be dropped from $\times_{(s,t)}$. Detailed proofs can be found in the full paper [8].

From an algebraic point of view we have a cpo with three continuous operations, application \diamond , union \star and product, \times . These operations obey the following laws:

1. \star satisfies:

(a) *Upper semi lattice axioms:*

- $d \star d = d$
- $d \star e = e \star d$
- $(d \star e) \star f = d \star (e \star f)$

(b) *Linearity of application:* $(d \star e) \diamond f = (d \diamond f) \star (e \diamond f)$

2. \times satisfies:

(a) $- \times - = -$

(b) $d \times e = e \times d$

(c) $(d \times e) \times f = d \times (e \times f)$

(d) $(d \times -) \star - = d \times -$

(e) *Linearity of product:* $d \times (e \star f) = (d \times e) \star (d \times f)$

(f) $[(d \star - \neq d) \wedge (e \star - \neq e)] \Rightarrow$
 $(d \times e) \diamond f = ((d \diamond f) \times e) \star (d \times (e \diamond f))$

3. $d \sqsubseteq [e \uplus -] \Rightarrow d \uplus - = d$

We now have enough structure to give semantics to the fragment of the language that we are considering. The following definition uses the familiar environment mechanism. The function Gr is the isomorphism from $\overline{P}(D)$ into D .

- $\llbracket x \rrbracket \rho = \rho(x)$
- $\llbracket \lambda x.M \rrbracket \rho = Gr(d \mapsto \llbracket M \rrbracket \rho[x \mapsto d])$
- $\llbracket \lambda \langle x_1, x_2 \rangle M \rrbracket \rho = Gr(\star[(d_1 \mapsto Gr(d_2 \mapsto \llbracket M \rrbracket \rho[x_1 \mapsto d_1, x_2 \mapsto d_2])), (d_1 \mapsto Gr(d_2 \mapsto \llbracket M \rrbracket \rho[x_2 \mapsto d_1, x_1 \mapsto d_2]))])$
- $\llbracket MN \rrbracket \rho = \llbracket M \rrbracket \rho \diamond \llbracket N \rrbracket \rho$
- $\llbracket M|N \rrbracket \rho = \llbracket M \rrbracket \rho \times \llbracket N \rrbracket \rho$

5 Adequacy

In this section, we describe the relationship between $-$ and non-termination in the calculus. This is, of course, crucial if our mathematical model is to say anything interesting about computation. Our model has the following adequacy properties.

Theorem 2. Let M be a closed term. Then, $\neg(M \Downarrow^{may}) \Rightarrow \llbracket M \rrbracket \rho = -$

Theorem 3. Let M be a closed term. Then, $\neg(M \Downarrow^{must}) \Rightarrow \llbracket M \rrbracket \rho \uplus - = \llbracket M \rrbracket \rho$

Theorem 4. $\llbracket M \rrbracket \rho \sqsubseteq \llbracket N \rrbracket \rho \Rightarrow M \preceq_b N$ for closed terms.

The proof superficially resembles the proof of adequacy in the setting of the lazy lambda calculus [1, 14]. The details, however, are rather more intricate than that situation as we have to deal with many possible reduction sequences; with indeterminacy in the calculus one cannot have a deterministic evaluation strategy. The detailed proof appears in the full paper [8]. In this version, we restrict ourselves to an overview of the proof.

We first introduce a labelled calculus and show that it is strongly normalizing. We then consider a reduction strategy \rightarrow_ω . We show that any \rightarrow_ω reduction in the labelled calculus can be mimicked in the γ -calculus. We then define a semantics for the labelled calculus in terms of approximable models [14, 20] equipped with extra structure to handle indeterminacy and concurrency and show that the meaning of a completely labelled term is less than the “union” of the meanings of all terms derived from one step \rightarrow_ω reductions. Because the fully labelled calculus is strongly normalizing and reduction is finitely-branching, we can classify all the “normal forms” that might exist after a fully labelled term is reduced. We can also show that the meaning of a term in the γ -calculus is given by the least upper bound of the meanings of the completely labelled terms derived from it. If we have a term M that never terminates, i.e. $\neg(M \Downarrow^{may})$, we can inspect all the terms that arise from reducing all its completely labelled versions and show that they all denote $-$. Thus the original term itself must have meaning bottom.

A similar but slightly more subtle argument is used for the “must converge” case. Suppose that we have a term, M , satisfying $\neg(M \Downarrow^{must})$. Reductions in the γ -calculus cannot be mimicked completely in the labelled version. However, if we examine a divergent reduction sequence of M and attempt to mimic it in the labelled calculus, we reach a point where the head redex has label 0. At this point we know that the meaning of the original term must “contain” \perp .

6 Conclusions and Future Work

The work in this paper represents part of a growing interest in higher-order process calculi. We feel that it is a significant achievement of Boudol’s to describe a calculus that can be given a pleasing mathematical model and yet express concurrency and abstraction.

There are other related calculi [13, 19, 4] and also the label passing calculi of Milner and his co-workers, studied independently by Engberg and Nielsen [6]. Though these systems are theoretical there are other closely related systems, in varying stages of formal analysis, that are actually implemented and are being used in experiments. The most interesting of these is Reppy’s calculus that incorporates events as first-class entities in Standard ML [16]. Though our work does not directly bear on these activities it does indicate that these ideas are ripe for an intensive study.

We plan to extend our model to the full calculus. We would also like to understand what it takes to make the calculus fully abstract. Given that the language has concurrency built into it, one might expect that one would get full abstraction by adding a simple convergence tester. Unlike the case of the lazy lambda calculus where one needed a parallel convergence tester. This, however, seems unlikely though we do not yet have any definitive answers as yet. We also plan to understand the structure of the powerdomain more clearly. Finally, we would like to relate these semantical investigations to the other formalisms cited.

Acknowledgements

We would like to thank B. Bloom and V. Shanbhogue for reading earlier versions of this paper and suggesting several improvements.

References

- [1] S. Abramsky and C. H. L. Ong. Full abstraction in the lazy lambda calculus. *Submitted to Information and Computation*, 1988.
- [2] H. P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. Studies in Logic. North-Holland, Amsterdam, revised edition edition, 1984.
- [3] G. Berry. Some syntactic and categorical constructions of lambda-calculus models. Technical Report 80, Institute National de Recherche en Informatique et en Automatique (INRIA), 1981.

- [4] G. Berry and G. Boudol. The chemical abstract machine. In *Proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages*, 1990.
- [5] G. Boudol. Towards a lambda-calculus for concurrent and communicating systems. In J. Diaz, editor, *TAPSOFT 89, Lecture Notes in Computer Science 351*, pages 149–161. Springer-Verlag, 1989.
- [6] U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. DAIMI PB-208, Aarhus University, 1986.
- [7] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
- [8] R. Jagadeesan and P. Panangaden. A domain-theoretic model for a higher-order process calculus. In *Proceedings of The International Conference on Automata, Languages and Programming*, pages 181–194. Springer-Verlag, July 1990. LNCS 443, Cornell TR 89-1058.
- [9] B. Jonsson. A fully abstract trace model for dataflow networks. In *Proceedings of the Sixteenth Annual ACM Symposium On Principles Of Programming Languages*, 1989.
- [10] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74*, pages 993–998. North-Holland, 1977.
- [11] J. Kok. A fully abstract semantics for dataflow nets. In *Proceedings of Parallel Architectures And Languages Europe 1987*, pages 351–368, Berlin, 1987. Springer-Verlag.
- [12] R. Milner, J. Parrow, and D. Walker. Mobile processes. Technical report, University of Edinburgh, 1989. Draft.
- [13] F. Nielson. The typed lambda-calculus with first-class processes. In *PARLE89, Lecture Notes in Computer Science 366*, pages 357–373, 1989.
- [14] C. H. L. Ong. *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*. PhD thesis, Imperial College of Science and Technology, 1988.
- [15] G. D. Plotkin. A powerdomain construction. *SIAM Journal of Computing*, 5(3):452–487, 1976.
- [16] J. H. Reppy. Synchronous operations as first-class values. In *Proceedings of the SIGPLAN conference on Programming language design and implementation*, 1988.
- [17] J. R. Russell. Full abstraction for nondeterministic dataflow networks. In *Proceedings of the 30th Annual Symposium of Foundations of Computer Science, Lecture Notes in Computer Science 442*, pages 170–177, 1989.
- [18] M. B. Smyth and G. D. Plotkin. The category theoretic solution of recursive domain equations. *Siam Journal of Computing*, 11(4), 1982.
- [19] B. Thomsen. A calculus of higher-order communicating systems. In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Programming Languages*, 1988.

- [20] C. P. Wadsworth. The relation between computational and denotational properties for scott's d_∞ models of the λ -calculus. *SIAM J. Computing*, 5:488–521, 76.