A Complete Order-theoretic Model for the Algebra of Communicating Processes

Peter M.W. Knijnenburg
Dept. of Computer Science, Leiden University,
Niels Bohrweg 1, 2333 CA Leiden, the Netherlands. E-mail: peterk@cs.leidenuniv.nl

Abstract

In this paper an order-theoretic denotational semantics for a small programming language is defined. The main result is that this denotational semantics is a sound and complete model for the equational theory of the language. As an immediate corollary we obtain that two process terms are bisimular if and only if they are denotationally equal.

1 Introduction

Much research in the study of the semantics of concurrent programming language focusses on so-called *uniform languages*. These languages are defined by a collection of elementary programs or atomic actions, and a number of program constructors like sequential composition ';' and choice '+'. The languages are called 'uniform' since these atomic actions are not further specified. The operational meaning of such programs is usually given by menas of a *transition system* which formalizes the elementary steps the program can take. A transition system is usually specified by means of a *Structured Operational Semantics (SOS)* system, in the style of Plotkin [Plo81b]. A transition system immediately gives rise to the notion of a process graph: the tree obtained by pasting transitions together. Obviously, two programs can be textually different but not distinguishable by any context, like a and $a + a$. Milner [Mil80] proposed to identify *bisimilar* processes. Bisimilarity is a congruence for a large class of transition systems [GV92]. In this way one obtains an equational theory of programs. These equational theories are called process algebras. Examples include Milner's *CCS* [Mil80, Mil89], Hoare's *CSP* [Hoa85] and Bergstra and Klop's *ACP* [BW91, BK84, BK85].

A denotational semantics is an interpretation of the atomic actions and program constructors of the language in some semantic domain. In order to deal with recursion, the domains should allow the construction of (least) fixed points. One therefore considers domains that are complete partial orders [Sco76, Plo81a], or complete metric spaces [dBZ82]. If one wants to model uniform languages, one typically uses a domain specified by a recursive domain equation like

$$\mathbf{P} \cong \mathcal{P}^*(\{\delta\}_\perp + A_\perp + A_\perp \times \mathbf{P})$$

Here A is the set of atomic actions and $\delta \notin A$ is a special constant coding the denotation of the deadlocked process. These kinds of domain equations can be found in numerous places in the literature, for example [Abr91a, dBMOZ88, dBR92, HP79]. The domain equation above is used in this paper. Intuitively, the domain $\mathbf{P}$ codes finitely branching, possibly infinite trees labeled by atomic actions.

Since both denotational and bisimulation semantics for a particular language are based on trees labeled by elementary actions, we expect that they are closely connected. In this paper we make this connection precise in one particular case. We study a small and well-known concurrent programming language: denotational models for (subsets of) it, or closely related languages, have been formulated in [Abr91a, dBMOZ88, HP79, MM79, Rut90]. We give a denotational model based on the models given in those papers. The language is the term-language of the process algebra *ACP* [BW91, BK84, BK85].

We present a denotational model for the language based on the models given in the papers cited above. The main result of this paper is that the denotational model is a sound and complete model for the process algebra *ACP*. As an immediate corollary, we obtain that that two process terms are bisimular if and only if they are denotationally equal. This follows from that well-known fact that the equational theory of *ACP* precisely axiomatizes bisimularity. Hence another contribution of this paper is an approach to relate denoatatinal models with bisimulation via an equational theory.

Recently, a number of papers have appeared that study the connections between Labeled Transition Systems, Structured Operational Semantics and denotational semantics in a general framework. We give a brief overview of other results in this area. Abramsky has studied the relationship between Labeled Transition Systems and denotational semantics in a logical framework [Abr91a]. He has defined an order-theoretic domain of synchronization trees and a logic for transition systems, and shows that one is the Stone dual of the other. In fact, this paper is part of a much larger programme in which domains and certain 'domain logics' are related by Stone duality [Abr91b]. Using this duality result, Abramsky is able to synthesize domain theory, the theory of concurrency and systems behavior based on operational semantics, and logics of programs. The domain of bisimulation proposed by Abramsky in [Abr91a] is closely related to our domain $\mathbf{P}$. The semantics proposed in this paper maps programs to an inclusive subset $\mathbf{P}^*$ of $\mathbf{P}$. We can show [Kni93c, Kni93b] that $\mathbf{P}^*$ is isomorphic to the domain considered by Abramsky.

Groote and Vaandrager [GV92] have shown that for a wide class of SOS systems (including the systems in the GSOS format) bisimulation equivalence is a congruence. Closely related is the result by Aceto, Bloom and Vaandrager [ABV92] which shows that SOS systems in the GSOS format *induce* an equational theory which precisely characterizes bisimilarity. Hence this induced equational theory can be used to relate denotational equality and bisimularity in the same way as discussed in this paper.

Rutten has shown that SOS systems in the *positive* GSOS format (that is, without negative premisses) induce denotational models such that two programs are bisimilar if and only if they have the same denotation [Rut90]. This work has been carried out in the metric framework. Horita has obtained similar results [Hor89]. Recently, Rutten was able to strengthen his results to a larger class of SOS systems in the framework of non-well-founded sets [Rut92]. He is able to define a denotational semantics for the original language and show that this semantics characterizes bisimularity. Rutten and Turi have shown that recursive domain equations give rise to final coalgebras (for the functor corresponding to the domain equation) in the categories of complete partial orders, complete metric spaces and non-well-founded sets [RT93]. They show how the functor appearing in the domain equation gives rise to a notion of observation. They describe Labeled Transition Systems in this setting and show how a denotational semantics can be obtained that has the property that two programs are bisimilar if and only if they have the same denotation.

The main tool we employ in defining the semantic operators in the interpretation of $\mathcal{L}$ is a typed lambda calculus. The category **Cpo** of complete partially ordered sets with continuous maps is cartesian closed [LS86]. Hence it has a typed lambda calculus as its internal language. Any continuous function occurs as a constant in this language. Furthermore, any typed lambda term in the language has an interpretation as a continuous function (arrow) in the category. This implies that we only need to identify a small set of primitive continuous functions in order to be able to define complex functions by typed lambda terms. These complex functions are then *by definition* continuous and we can prove properties of them using the well-understood language of the typed lambda calculus.

This paper is organized as follows. In section 2 the syntax of the language and the bisimulation semantics is given. In section 3 we develop the necessary domain theory. In section 4 we give the definition of the domain **P** and list some of its properties. The latter two sections are self-contained and give a small overview of the domain theory needed in this paper. In section 5 we present our model and show that it provides a sound interpretation for the equational theory of *ACP*. In section 6 we show that it is sound and complete. Finally, in section 7 we discuss the results obtained in this paper.

Acknowledgement I would like to thank the Amsterdam Concurrency Group, headed by Jaco de Bakker, and the Utrecht Formal Models Group for stimulating discussions. In particular, I like to thank Joost Kok for critically reading draft versions of the paper. I also like to thank Wim Hesselink for valuable comments.

2 Syntax and equational theory of the language

In this section we introduce the language that we study in this paper and review the relevant equational theory of this language. For a fuller account of this equational theory, consult [BK84, BK85, BW91].

First we define the collection of finite, recursion-free terms $\mathcal{T}_{\mathrm{f}}$. Formally, $\mathcal{T}_{\mathrm{f}}$ is given by the following grammar:

$$s \ ::= \ a \mid \delta \mid (s;s) \mid (s+s) \mid (s \mathbin{\|\!\_} s) \mid (s|s) \mid (s \parallel s) \mid \partial_H(s) \mid x$$

where $x \in Var$, $a \in A$ and $H \subseteq A$. Here A is a (countable) set of constants or *atomic actions*, with typical element a, and Var is a (countable) set of variables, with typical element x. The collection of *closed terms* (that is, with no occurrence of a variable $x \in Var$) is denoted by $\mathcal{L}_{\mathrm{f}}$. Terms in $\mathcal{L}_{\mathrm{f}}$ are also called *programs*. In the sequel we are primarily interested in programs. The collection of all terms is only introduced in order formulate an equational theory (the axioms below use terms that are not closed).

The intuitive reading of the function symbols is the following: δ is a special constant that always deadlocks, ; denotes sequential composition, + denotes choice, $\parallel$ denotes merge, and $\mathbin{\|\!\_}$ and $|$ are left-merge and communication-merge, respectively. The left-merge operator is an auxillary operator needed to give a finite axiomatization of the merge operator, see [Mol90]. It acts like the merge-operator, but always performs an action from its left-hand side argument first. ∂_H is a unary function symbol for each finite subset $H \subseteq A$. It is the *encapsulation operator* that prevents actions in H to be visible outside its scope and blocks synchronization (using actions in H) with the environment. We assume a function $\gamma : (A \cup \{\delta\}) \times (A \cup \{\delta\}) \to (A \cup \{\delta\})$ that is commutative, associative and has δ as zero, that is, $\gamma(a,\delta) = \gamma(\delta,a) = \gamma(\delta,\delta) = \delta$ for all $a \in A$. This function encodes the basic communication between any actions $a, b \in A$: if $\gamma(a,b) = c \neq \delta$ then we say that a and b can synchronize. The synchronous execution of a and b is then regarded as the execution of c. If $\gamma(a,b) = \delta$ then a and b cannot synchronize and any attempt to synchronize them results in deadlock.

The axioms of the equational theory of ACP are given in Table 1 (see also [BK84, Vaa89]). In Table 1, axioms $A1-7$ are the axioms of Basic Process Algebra with Deadlock. Axiom CF relates the communication between elementary actions a and b to the given function γ. Axioms $CM1-9$ are the axioms dealing with parallel composition. In particular, they state that the parallel composition of two processes $s_1 \parallel s_2$ performs an action by either choosing one of its arguments and performing an action of that argument, or by synchronizing. Axioms $D1-4$ deal with the Encapsulation operator. Axioms $SC1-6$ are called 'Standard Concurrency Axioms'. These last axioms are not always included in the theory of ACP, since one can prove that they hold for all terms in $\mathcal{L}_{\mathrm{f}}$ [BW91, BK85]. Since they do not hold for arbitrary terms, we have included them in the present axiom system We have omitted the usual axioms for equality, like reflexivity and substitution (consult [vD83]).

$x + y = y + x$	$A1$	$\partial_H(a) = a$ if $a \notin H$	$D1$
$x + (y + z) = (x + y) + z$	$A2$	$\partial_H(a) = \delta$ if $a \in H$	$D2$
$x + x = x$	$A3$	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	$D3$
$(x + y); z = x; z + y; z$	$A4$	$\partial_H(x; y) = \partial_H(x); \partial_H(y)$	$D4$
$(x; y); z = x; (y; z)$	$A5$		
$x + \delta = x$	$A6$		
$\delta; x = \delta$	$A7$		
$a \vert b = \gamma(a, b)$	CF		
$x \parallel y = x \mathbin{\|\!\_} y + y \mathbin{\|\!\_} x + x \vert y$	$CM1$	$(x \mathbin{\|\!\_} y) \mathbin{\|\!\_} z = x \mathbin{\|\!\_} (y \parallel z)$	$SC1$
$a \mathbin{\|\!\_} x = a; x$	$CM2$	$(x \vert y) \mathbin{\|\!\_} z = x \vert (y \mathbin{\|\!\_} z)$	$SC2$
$(a; x) \mathbin{\|\!\_} y = a; (x \parallel y)$	$CM3$	$x \vert y = y \vert x$	$SC3$
$(x + y) \mathbin{\|\!\_} z = x \mathbin{\|\!\_} z + y \mathbin{\|\!\_} z$	$CM4$	$x \parallel y = y \parallel x$	$SC4$
$(a; x) \vert b = (a \vert b); x$	$CM5$	$x \vert (y \vert z) = (x \vert y) \vert z$	$SC5$
$a \vert (b; x) = (a \vert b); x$	$CM6$	$x \parallel (y \parallel z) = (x \parallel y) \parallel z$	$SC6$
$(a; x) \vert (b; y) = (a \vert b); (x \parallel y)$	$CM7$		
$(x + y) \vert z = x \vert z + y \vert z$	$CM8$		
$x \vert (y + z) = x \vert y + x \vert z$	$CM9$		

Table 1: Axioms for ACP

Another important result of the equational theory is that every program s has a *normal form*. The collection $\mathcal{N} \subseteq \mathcal{L}_\mathrm{f}$ (modulo axioms $A1$ and $A2$) of normal forms is the smallest set closed under

- $\delta \in \mathcal{N}$ and $A \subseteq \mathcal{N}$
- for all $a \in A$ and $s \in \mathcal{N}$, $a; s \in \mathcal{N}$
- for distinct $s_1 \in \mathcal{N}, \ldots, s_n \in \mathcal{N}$, where $n > 1$ and every s_i is distinct from δ, $s_1 + \cdots + s_n \in \mathcal{N}$.

For any $s \in \mathcal{L}_\mathrm{f}$ there exists a $s' \in \mathcal{N}$ such that $\vdash s = s'$. Moreover, s' is unique up to the ordering and bracketing of the summands. Hence we can choose a function $\mathcal{NF} : \mathcal{L}_\mathrm{f} \to \mathcal{N}$ assigning to each term (one of) its normal form. We have the following proposition [BW91].

Proposition 2.1 *For all $s, t \in \mathcal{L}_f$, $\vdash s = t$ if and only if $\vdash \mathcal{NF}(s) = \mathcal{NF}(t)$.*

We now discuss how to add recursion to the language. First of all, we assume a (countable) collection $PVar$ of procedure variables or names, with typical element X. The collection of terms obtained by adding these procedure variables as constants is denoted by $\mathcal{T}$. The collection of all closed terms, or programs, is denoted by $\mathcal{L}$. Procedure variables are identified with their body. For technical reasons we introduce the following sublanguage consisting of *guarded statements* $\mathcal{L}_{\mathrm{g}} \subseteq \mathcal{L}$ for these procedure bodies:

$$g \;\; (\in \mathcal{L}_{\mathrm{g}}) \;\; ::= \;\; a \mid \delta \mid (g; s) \mid (g_1 + g_2) \mid (g_1 \parallel g_2) \mid (g \mathbin{\lfloor\!\lfloor} s) \mid (g_1 | g_2) \mid \partial_H(g)$$

Intuitively, a statement is guarded if every occurrence of a procedure variable is preceded by at least one elementary action. A function $d : PVar \to \mathcal{L}_{\mathrm{g}}$ is called a *declaration.* Hence the body of a procedure is always a guarded statement. We sometimes emphasis this by saying that the recursion is guarded.

Since procedures may be recursive, they need not be finite. For instance, if we assume $d(X) = a; X$, then X is the program that executes infinitely many atomic actions a. This means that programs $s \in \mathcal{L}$ do not always possess a normal form in $\mathcal{N}$ which consists of finite terms. Moreover, assuming $d(Y) = a; Y$, we have no way of proving that $X = Y$ using only the equations given above.

To overcome these problems, one introduces the following *projection operators* π_n, for $n \geq 1$, which allow n actions to be executed. We define π_n inductively on the set of normal forms as

$$\begin{array}{rcl}
\pi_n(a) & = & a \\
\pi_n(\delta) & = & \delta \\
\pi_1(a; s) & = & a \\
\pi_{n+1}(a; s) & = & a; \pi_n(s) \\
\pi_n(s_1 + \cdots + s_m) & = & \pi_n(s_1) + \cdots + \pi_n(s_m)
\end{array}$$

Note that the projection of a normal form is again a normal form. These projections extend to functions $\pi_n : \mathcal{L} \to \mathcal{N}$ by the following procedure. Let s be an arbitrary term in $\mathcal{L}$.

1. Define inductively the term $s^{(n)}$ as follows. $s^{(0)} \equiv s$, and given $s^{(n)}$, define $s^{(n+1)}$ to be the term obtained by replacing every procedure variable X by its body $d(X)$.
2. Obtain the normal form $\mathcal{NF}(s^{(n)})$ by considering each procedure variable X occurring in $s^{(n)}$ as a (new) constant action.
3. Put $\pi_n(s) = \pi_n(\mathcal{NF}(s^{(n)}))$.

By guardedness of the recursion this is well-defined.

Using these projection operators we can now formulate an infinitary rule to deal with infinite programs. This rule is called the *Approximation Induction Principle* (AIP). Consult [BW91] for a more comprehensive discussion of this rule.

$$\frac{\{\pi_n(x) = \pi_n(y) : n < \omega\}}{x = y}$$

Note that this rule can only be used if we substitute closed terms for x and y (otherwise, the projections $\pi_n(x)$ are not defined). In section 5 we show that the syntactic 'finite approximations' $\pi_n(s)$ to a program s are closely related to the finite approximations to the denotation of s in the order-theoretic sense.

3 Domain Theory

In this section we give the mathematical preliminaries on which this work is based. This section introduces the various constructions and notations we use in the sequel. For a more complete treatment of the theory, consult [GS90, Plo81a].

Partial orders. A tuple $(D, \sqsubseteq)$ where D is a set and $\sqsubseteq\ \subseteq D \times D$ is a relation on D, is called a *partial order* if $\sqsubseteq$ is reflexive, transitive and anti-symmetric. We assume that each partial order has a distinguished element $\perp$ that is least with respect to the ordering relation, that is, $\perp \sqsubseteq d$ for all $d \in D$. Given a partial order $(D, \sqsubseteq)$, we call a subset $\{x_i : i < \omega\}$ a *chain* if $x_i \sqsubseteq x_{i+1}$ for all $i < \omega$. In this case we write $(x_i)_i$ for the subset. An element $d \in D$ is an *upperbound* for a chain $(x_i)_i$ if $x_i \sqsubseteq d$ for all i. It is a *least upperbound* (lub) if $d \sqsubseteq d'$ for any upperbound d'. We write $\bigsqcup_i x_i$ for the unique least upperbound of a chain $(x_i)_i$, if it exists. A partial order $(D, \sqsubseteq)$ is called *complete* (or a *cpo*) if it has least upperbounds for all chains.

The notion of chain is fundamental in the study of semantics of programming languages. Essentially, the meaning of a (recursive) program is viewed as the lub of (inductively defined) approximations to it. The approximations are all in some sense "finite". We formalize this as follows. An element $d \in D$ for some cpo D is called *finite* if $d \sqsubseteq \bigsqcup_i x_i$ implies $d \sqsubseteq x_k$ for some k. We denote the collection of all finite elements of some cpo D by $K(D)$. Given our goal of defining a semantics, it is natural to restrict attention to cpo's that are completely determined by their finite elements. That is, for all $x \in D$ we wish there to exist a chain $(x_i)_i \subseteq K(D)$ such that $x = \bigsqcup_i x_i$. These cpo's are called *algebraic*. They are called *ω-algebraic* if moreover this collection of finite elements is a countable set. We will use the term 'domain' for an ω-algebraic cpo.

Functions. The natural notion of function between sets with some structure is of course a function that preserves the structure. In our case, the function should at least preserve the ordering. That is, if $f : D \to E$ and $x \sqsubseteq y \in D$ then $f(x) \sqsubseteq f(y) \in E$. We call these functions *monotonic*. Functions f such that $f(\perp) = \perp$ are called *strict*. The next restriction that we impose on functions is that $f(\bigsqcup_i x_i) = \bigsqcup_i f(x_i)$. These functions are called *continuous*.

Since we are working with domains in which all elements arise as lub's of chains of finite elements, the continuous functions $f : D \to E$ stand in a one-to-one correspondence with monotonic functions $f' : K(D) \to E$ (*c.f.* [Plo81a]). This means that every continuous function is completely determined by its action on the finite elements. This fact enables us to define a continuous function $f : D \to E$ by specifying a monotonic function $f' : K(D) \to E$.

Proposition 3.1 *Let D and E be domains. Let $f : K(D) \to E$ be monotonic and let $g : D \to E$ be monotonic and continuous. Define $\uparrow f : D \to E$ by*

$$\uparrow f(x) = \bigsqcup f(x_i)$$

where $(x_i)_i \subseteq K(D)$ is some chain such that $x = \bigsqcup_i x_i$. Then

1. *$\uparrow f$ is well-defined and continuous.*
2. *$f = (\uparrow f) \restriction K(D)$.*
3. *$g = \uparrow (g \restriction K(D))$.*

Given a function $f : D \to D$, a fixed point of f is a value $d \in D$ such that $f(d) = d$. All continuous functions have fixed points. The least fixed point of a continuous f is given by $\bigsqcup_n f^n(\perp)$. For all D, the function $fix_D : [D \to D] \to D$ that yields this least fixed point is a continuous function.

Domain constructions. First of all, observe that we can turn each countable set X into a domain $X_\perp$ by adjoining a new least element $\perp$ and stipulating that $\perp \sqsubseteq x$ and $x \sqsubseteq x$ for all $x \in X$. Cpo's of this form are called *flat*. Every set theoretic function $f : X \to Y$ can be extended to a continuous function $f_\perp : X_\perp \to Y_\perp$ by defining $f_\perp(\perp) = \perp$ and $f_\perp(x) = f(x)$ for $x \in X$.

Let D and E be domains. We define the following constructs yielding new domains:

- $D \times E$ is the cartesian product of D and E. The underlying set is

  $$\{\langle x, y\rangle : x \in D, y \in E\}$$

  The order is given by $\langle x_1, y_1\rangle \sqsubseteq \langle x_2, y_2\rangle$ iff $x_1 \sqsubseteq x_2$ and $y_1 \sqsubseteq y_2$. Its bottom element is $\langle \perp, \perp\rangle$.

- $D + E$ is the sum of D and E. The underlying set is

  $$\{\langle 0, x\rangle : x \in D \setminus \{\perp\}\} \cup \{\langle 1, y\rangle : y \in E \setminus \{\perp\}\} \cup \{\perp\}$$

  This is just the counterpart of the disjoint union of ordinary sets. For $x, y \in D+E$, the order is given by $x \sqsubseteq y$ iff $x \equiv \perp$ or $x \equiv \langle i, x'\rangle, y \equiv \langle i, y'\rangle$ and $x' \sqsubseteq y'$ $(i = 0, 1)$. Note that $D + E$ is the coproduct of D and E with respect to strict functions.

- One can generalize $+$ to arbitrary finite sums. We denote this by $D_0 + \cdots + D_{n-1}$.

If D and E are domains, then so are $D \times E$ etc. All constructions come with special functions to and from them. We define

- projections $\pi : D \times E \to D$ and $\pi' : D \times E \to E$ given by $\pi\langle x, y\rangle = x$ and $\pi'\langle x, y\rangle = y$, respectively.

- if $f : A \to D$ and $g : A \to E$ then $\langle f, g\rangle : A \to D \times E$ is given by $\langle f, g\rangle(a) = \langle f(a), g(a)\rangle$. Note that $\pi \circ \langle f, g\rangle = f$ and $\pi' \circ \langle f, g\rangle = g$.

- inclusions $in_0 : D \to D + E$ and $in_1 : E \to D + E$ given by $in_i(\perp) = \perp$ and $in_i(x) = \langle i, x\rangle$.

- sum projections $out_0 : D + E \to D$ and $out_1 : D + E \to E$ given by

$$out_0(x) = \begin{cases} \perp & \text{if } x \equiv \perp \\ \perp & \text{if } x \equiv \langle 1, y'\rangle \\ x' & \text{if } x \equiv \langle 0, x'\rangle \end{cases}$$

  and likewise for out_1.

- sum discriminators $is_0 : D + E \to \mathbf{T}$ and $is_1 : D + E \to \mathbf{T}$ given by

$$is_0(x) = \begin{cases} \perp & \text{if } x \equiv \perp \\ \mathbf{t} & \text{if } x \equiv \langle 0, x'\rangle \\ \mathbf{f} & \text{if } x \equiv \langle 1, x'\rangle \end{cases}$$

  and likewise for is_1. Here we have used the flat domain of truth values $\mathbf{T} = \{\perp, \mathbf{t}, \mathbf{f}\}$.

- all functions can be extended to finite sums.

Finally we review the construction of the Plotkin powerdomain $\mathcal{P}^*(D)$. It is more difficult than the previous ones, and a more detailed exposition can be found in [Plo79, Smy78, Kni93a]. We start with the collection $\mathcal{F}(D)$ of all finite sets of finite elements of D. These will act as the finite elements of $\mathcal{P}^*(D)$. We order $\mathcal{F}(D)$ by putting

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

This order is called the Egli-Milner order. $(\mathcal{F}(D), \sqsubseteq_{EM})$ is a pre-ordered set, hence we can form its completion to get a domain (see [Kni93a]). This domain is by definition the powerdomain of D. A suitable representation uses the following operation (*c.f.* [Plo81a, Kni93a]).

$$\mathcal{C}\ell(X) = \{\bigsqcup x_i \subseteq K(D) : \exists x \in X.x \sqsubseteq \bigsqcup x_i \;\&\; \forall i \exists x' \in X.x_i \sqsubseteq x'\}$$

It is easy to check that $\mathcal{C}\ell$ indeed is a set theoretical closure operation. Furthermore we have $\mathcal{C}\ell(X) = Con(X)$ for all $X \in \mathcal{F}(D)$ where $Con(X) = \{y : \exists x_1, x_2 \in X.x_1 \sqsubseteq y \sqsubseteq x_2\}$ is the convex closure operator. Defining

$$Up(X_i)_i = \{\bigsqcup x_i : x_i \in X_i\}$$

for a chain $(X_i)_i \subseteq \mathcal{F}(D)$, the powerdomain then consists of all sets $\mathcal{C}\ell(Up(X_i)_i)$. Unfortunately, the ordering of these sets is in general no longer Egli-Milner, but becomes the Plotkin order. For details, consult [Plo81a, Smy78, Kni93a]. We will always be working with finite elements, and these are Egli-Milner ordered.

- For $f : D \to E$ we define $\mathcal{P}^*f : \mathcal{P}^*(D) \to \mathcal{P}^*(E)$ by
$$\mathcal{P}^*f(X) = \mathcal{C}\ell\{f(x) : x \in X\}$$
- For $f : D_1 \times D_2 \to E$ we define $f^\dagger : \mathcal{P}^*(D_1) \times D_2 \to \mathcal{P}^*(E)$ by
$$f^\dagger(X, y) = \mathcal{C}\ell\{f(x, y) : x \in X\}$$
- For $f : D_1 \times D_2 \to E$ we define $f^\ddagger : \mathcal{P}^*(D_1) \times \mathcal{P}^*(D_2) \to \mathcal{P}^*(E)$ by
$$f^\ddagger(X_1, X_2) = \mathcal{C}\ell\{f(x_1, x_2) : x_1 \in X_1, x_2 \in X_2\}$$
- We have a continuous function $\uplus : \mathcal{P}^*(D) \times \mathcal{P}^*(D) \to \mathcal{P}^*(D)$ given by
$$X \uplus Y = \mathcal{C}\ell(X \cup Y)$$
- We have a continuous function $\{\!|\cdot|\!\} : D \to \mathcal{P}^*(D)$ given by $\{\!|x|\!\} = \{x\}$

The constructions $(\cdot) \times (\cdot)$, $\mathcal{P}^*(\cdot)$ etc., are all continuous. That is, for a chain of functions $(f_i)_i$ we have $g \times \bigsqcup_i f_i = \bigsqcup_i (g \times f_i)$ etc. In particular this means that one can solve recursive domain equations involving these constructors (*c.f.* [Plo81a, SP82]). Also, the derived operations $(\cdot)^\dagger$ and $(\cdot)^\ddagger$ are continuous.

A function $f : \mathcal{P}^*(D) \to \mathcal{P}^*(E)$ is called *linear* if $f(X \uplus Y) = f(X) \uplus f(Y)$ for all $X, Y \in \mathcal{P}^*(D)$. It is known that $\mathcal{P}^*f$, $(f)^\dagger$ and $(f)^\ddagger$ are linear [Plo81a, Plo79].

We need the following continuous function **if** $(\cdot)$ **then** $(\cdot)$ **else** $(\cdot) : \mathbf{T} \times D \times D \to D$ given by
$$\textbf{if } t \textbf{ then } d_1 \textbf{ else } t_2 = \begin{cases} \bot & \text{if } t = \bot \\ d_1 & \text{if } t = \mathbf{t} \\ d_2 & \text{if } t = \mathbf{f} \end{cases}$$

4 The Semantic Domain

In this section we give the definition of the domain $\mathbf{P}$ that underlies our order-theoretic denotational semantics for $\mathcal{L}$. $\mathbf{P}$ is defined as the least solution of the following reflexive equation
$$\mathbf{P} \cong \mathcal{P}^*(\{\delta\}_\bot + A_\bot + A_\bot \times \mathbf{P})$$
where $\delta \notin A$ is used to denote deadlock. For definiteness, let ϕ be the isomorphism between the left- and right-hand sides.

It is instructive to see how the solution $\mathbf{P}$ is obtained as we will need the construction in the sequel. For a full treatment of the theory of solving reflexive domain equations, see [Plo81a, SP82]. The following theory is taken from those papers. Briefly, we solve the equation in $\mathbf{Cpo}^E$, the category that has as objects cpo's and as arrows *embedding-projection pairs* $(k : D \to E, l : E \to D)$ satisfying $l \circ k = 1_D$ and $k \circ l \sqsubseteq 1_E$. Now $\mathbf{P}$ is the colimit of the sequence
$$P_0 = \{\bot\} \qquad P_{n+1} = \mathcal{P}^*(\{\delta\}_\bot + A_\bot + A_\bot \times P_n)$$

with as embedding-projection pairs the pairs $(i_n : P_n \to P_{n+1}, j_n : P_{n+1} \to P_n)$ given by

$$\begin{array}{rlrl} i_0 &= \lambda x.\bot & j_0 &= \lambda x.\bot \\ i_{n+1} &= \mathcal{P}^*(1_{\{\delta\}_\bot} + 1_{A_\bot} + 1_{A_\bot} \times i_n) & j_{n+1} &= \mathcal{P}^*(1_{\{\delta\}_\bot} + 1_{A_\bot} + 1_{A_\bot} \times j_n) \end{array}$$

Intuitively, i_n is the inclusion of P_n into P_{n+1} and j_n maps a set at nesting depth $n+1$ onto $\bot$. We write

$$i_{nm} = i_{m-1} \circ \cdots \circ i_{n+1} \circ i_n \quad \text{and} \quad j_{nm} = j_n \circ j_{n+1} \circ \cdots \circ j_{m-1}$$

The colimit comes equipped with functions $\alpha_n : P_n \to \mathbf{P}$ and $\beta_n : \mathbf{P} \to P_n$. These functions have the properties that

1. $\alpha_n \circ \beta_n \sqsubseteq \alpha_{n+1} \circ \beta_{n+1}$,
2. $\bigsqcup_n \alpha_n \circ \beta_n = 1_{\mathbf{P}}$,
3. $\alpha_n = \alpha_{n+1} \circ i_n$ and $\beta_n = j_n \circ \beta_{n+1}$.

Concretely, $\mathbf{P}$ consists of ω-indexed sequences

$$p \equiv \langle p_0, p_1, \ldots, p_n, \ldots \rangle$$

where $p_n \in P_n$ such that $p_n = j_n(p_{n+1})$. Then

$$\beta_n(\langle p_0, \ldots, p_n, \ldots \rangle) = p_n$$

$$\alpha_n(p) = \langle j_{0n}(p), \ldots, j_{(n-1)n}(p), p, i_{n(n+1)}(p), \ldots \rangle$$

As $\{\delta\}_\bot$ and $A_\bot$ are ω-algebraic, every P_n is ω-algebraic. In turn, $\mathbf{P}$ itself is ω-algebraic and its collection of finite elements is given by $\alpha_n(p)$ for $p \in K(P_n)$.

We now give a characterization of the continuous functions $f : \mathbf{P} \to \mathbf{P}$ in terms of functions $f_n : P_n \to P_n$. This characterization will enable us to derive properties of functions $f : \mathbf{P} \to \mathbf{P}$ by showing that these properties hold of certain (induced) functions $f^{(n)} : P_n \to P_n$ and invoking a limit argument. The latter task will be substantially easier since we are allowed to reason by inductive arguments. To our knowledge, this approach is new.

First of all, to each $f : \mathbf{P} \to \mathbf{P}$ we associate functions $f^{(n)} : P_n \to P_n$ by stipulating that $f^{(n)} = \beta_n \circ f \circ \alpha_n$ for each n. Observe that we have

$$\begin{array}{rcl} j_n \circ f^{(n+1)} \circ i_n &=& j_n \circ \beta_{n+1} \circ f \circ \alpha_{n+1} \circ i_n \\ &=& \beta_n \circ f \circ \alpha_n \\ &=& f^{(n)} \end{array}$$

We call a family of functions $\{f_n : P_n \to P_n : n < \omega\}$ *compatible* if $f_n = j_n \circ f_{n+1} \circ i_n$ and write $[f_n]_n$ for such a family. We say that a family of functions is *strongly* compatible if $f_n \circ j_n = j_n \circ f_{n+1}$. Note that if both $[f_n]_n$ and $[g_n]_n$ are strongly compatible then so is $[f_n \circ g_n]_n$.

Each $f_n : P_n \to P_n$ gives rise to a function $\bar{f_n} : \mathbf{P} \to \mathbf{P}$ given by $\bar{f_n} = \alpha_n \circ f_n \circ \beta_n$. For a compatible family $[f_n]_n$ we have

$$\begin{array}{rcl} \bar{f_n} & = & \alpha_n \circ j_n \circ f_{n+1} \circ i_n \circ \beta_n \\ & \sqsubseteq & \alpha_{n+1} \circ f_{n+1} \circ \beta_{n+1} \\ & = & \bar{f}_{n+1} \end{array}$$

Hence we may define $[f_n]_n^{\uparrow} = \bigsqcup_n \bar{f_n}$. This is a continuous function. Note that if $[f_n]_n$ is strongly compatible, then

$$[f_n]_n^{\uparrow} \langle p_0, p_1, p_2, \ldots \rangle = \langle f_0(p_0), f_1(p_1), f_2(p_2), \ldots \rangle$$

Proposition 4.1 *For all $f : \mathbf{P} \to \mathbf{P}$ and all compatible families $[f_n]_n$ we have*

1. *$[f^{(n)}]_n$ is compatible.*
2. *$[f_n]_n^{\uparrow}$ is continuous.*
3. *$[f^{(n)}]_n^{\uparrow} = f$.*
4. *$\forall n. \left([f_n]_n^{\uparrow}\right)^{(n)} = f_n$.*

Proof We have already proven 1 and 2 above. Next,

$$\begin{array}{rcl} [f^{(n)}]_n^{\uparrow} & = & \bigsqcup \alpha_n \circ \beta_n \circ f \circ \alpha_n \circ \beta_n \\ & = & \bigsqcup (\alpha_n \circ \beta_n) \circ f \circ \bigsqcup (\alpha_n \circ \beta_n) \\ & = & 1_{\mathbf{P}} \circ f \circ 1_{\mathbf{P}} \\ & = & f \\ \left([f_n]_n^{\uparrow}\right)^{(n)} & = & \beta_n \circ [f_n]_n^{\uparrow} \circ \alpha_n \\ & = & \beta_n \circ \bigsqcup (\alpha_m \circ f_m \circ \beta_m) \circ \alpha_n \\ & = & \bigsqcup (\beta_n \circ \alpha_m \circ f_m \circ \beta_m \circ \alpha_n) \\ & = & \bigsqcup (j_{nm} \circ f_m \circ i_{nm}) \\ & = & f_n \end{array}$$

$\square$

5 The Denotational Semantics

In this section we define the denotational semantics $[\![-]\!] : \mathcal{L} \to \mathbf{P}$ and prove that this semantics provides a model for the equational theory as given in section 2. To this end we define semantic functions corresponding to the syntactic constructors of the language. We define the semantic functions as least fixed points of certain higher-order functionals. The functionals themselves are defined using a small collection of primitive functions that we know to be continuous. Consequently, the resulting expressions are automatically continuous functions themselves. We prove several properties of the semantic functions and indicate the axioms of *ACP* that correspond to these properties.

5.1 Sequential composition

We want to define a function $\odot : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$. Intuitively, given two processes $p_1, p_2 \in \mathbf{P}$, the process $p_1 \odot p_2$ is obtained by attaching copies of p_2 to the endpoints of p_1. We shall define this function pointwise. That is, we first show how to attach such a copy of p_2 to each 'element' in p_1. We then collect all these results together to obtain $p_1 \odot p_2$. First define the higher order operator Ψ

$$\Psi : [\mathbf{P} \times \mathbf{P} \to \mathbf{P}] \to [(\{\delta\}_\perp + A_\perp + A_\perp \times \mathbf{P}) \times \mathbf{P} \to (\{\delta\}_\perp + A_\perp + A_\perp \times \mathbf{P})]$$

by

$$\begin{aligned} \Psi(f)(x,p) = \ & \mathbf{if}\ is_0(x)\ \mathbf{then}\ x \\ & \mathbf{else\ if}\ is_1(x)\ \mathbf{then}\ in_2(\langle out_1(x), p\rangle) \\ & \qquad \mathbf{else}\ in_2(\langle \pi(out_2(x)), f(\pi'(out_2(x)), p)\rangle) \end{aligned}$$

and $\tilde{\Psi} = \lambda f.\phi^{-1} \circ (\Psi_l(f))^\dagger \circ (\phi \times 1)$. Put $\odot = fix(\tilde{\Psi})$.

In the remainder of this section, we suppress the functions in_i and out_i for the sake of simplicity, trusting that the reader can provide them himself where needed. For instance, we will write $\{\!|a|\!\}$ instead of $\{\!|in_1(a)|\!\}$. This convention increases readability of the expressions considerably.

Lemma 5.1 *For all n, we have that $\odot^{(n)} = \odot_n$, where $\odot_n = (\tilde{\odot}_n)^\dagger$ using*

$$\tilde{\odot}_{n+1} : [\{\delta\}_\perp + A_\perp + A_\perp \times P_n] \times P_{n+1} \to [\{\delta\}_\perp + A_\perp + A_\perp \times P_n]$$

inductively given by

$$\begin{aligned} \tilde{\odot}_0 &= \lambda x, y.\ \perp \\ \tilde{\odot}_{n+1} &= \lambda x, y.\ \mathbf{if}\ is_0(x)\ \mathbf{then}\ x \\ & \qquad\quad \mathbf{else\ if}\ is_1(x)\ \mathbf{then}\ \langle x, j_n(y)\rangle \\ & \qquad\qquad \mathbf{else}\ \langle \pi(x), \pi'(x) \odot_n j_n(y)\rangle \end{aligned}$$

Proof First we observe that we can define α_n and β_n with the help of functions $\tilde{\alpha}_n$ inductively given by

$$\mathbf{if}\ is_0(x) \vee is_1(x)\ \mathbf{then}\ x\ \mathbf{else}\ \langle \pi(x), \alpha_n(\pi'(x))\rangle$$

putting $\alpha_n = \phi^{-1} \circ \mathcal{P}^*(\tilde{\alpha}_n) \circ \phi$. Likewise for β_n. Hence both α_n and β_n are linear.

We proceed by induction on n to show that $\odot^{(n)} = \odot_n$. The case $n = 0$ is trivial. For $n \geq 0$, we have $p_1 \odot^{(n+1)} p_2 = \beta_{n+1}(\alpha_{n+1}(p_1) \odot \alpha_{n+1}(p_2))$. This last function is the pointwise extension of

$$\begin{aligned} & \mathbf{if}\ is_0(x)\ \mathbf{then}\ x \\ & \mathbf{else}\ \ \mathbf{if}\ is_1(x)\ \mathbf{then}\ \langle x, \beta_n(\alpha_{n+1}(p_2))\rangle \\ & \qquad\ \mathbf{else}\ \langle \pi(x), \beta_n(\alpha_n(\pi'(x)) \odot \alpha_{n+1}(p_2))\rangle \end{aligned}$$

Since $\beta_n \circ \alpha_{n+1} = j_n$ and $\beta_n(p \odot \alpha_m(p')) = \beta_n(p \odot \alpha_n(j_{mn}(p')))$ for all n and $m \geq n$, we have that $\beta_n(\alpha_n(\pi'(x)) \odot \alpha_{n+1}(p_2)) = \beta_n(\alpha_n(\pi'(x)) \odot \alpha_n(j_n(p_2)))$. Hence the desired conclusion. $\square$

Corollary 5.2 $\odot^{(n)} \circ (j_n \times j_n) = j_n \circ \odot^{(n+1)}$.

Lemma 5.3 *For all* $p_1, p_2, p_3 \in \mathbf{P}$,

1. $(p_1 \odot p_2) \odot p_3 = p_1 \odot (p_2 \odot p_3)$ *(Axiom A5);*
2. $\{\!|\delta|\!\} \odot p_1 = \{\!|\delta|\!\}$ *(Axiom A6).*

Proof First we show that for all n, $(p'_1 \odot^{(n)} p'_2) \odot^{(n)} p'_3 = p'_1 \odot^{(n)} (p'_2 \odot^{(n)} p'_3)$ for all $p'_1, p'_2, p'_3 \in P_n$. The case $n = 0$ is trivial. For the induction step we argue as follows. The right hand side of the equation, considered as a function of p'_1, p'_2, p'_3, is the pointwise extension of

$$\begin{array}{l} \textbf{if } is_0(x) \textbf{ then } x \\ \textbf{else } \ \textbf{if } is_1(x) \textbf{ then } \langle x, j_n(p'_2 \odot^{(n+1)} p'_3)\rangle \\ \qquad\ \ \textbf{else } \langle \pi(x), \pi'(x) \odot^{(n)} j_n(p'_2 \odot^{(n+1)} p'_3)\rangle \end{array}$$

By properties of the compatible family $[\odot^{(n)}]_n$, we have that

$$\pi'(x) \odot^{(n)} j_n(p'_2 \odot^{(n+1)} p'_3) = \pi'(x) \odot^{(n)} (j_n(p'_2) \odot^{(n)} j_n(p'_3))$$

By induction hypothesis, the last expression equals

$$(\pi'(x) \odot^{(n)} j_n(p'_2)) \odot^{(n)} j_n(p'_3)$$

Now it is easy to see that the left-hand side is the pointwise extension of the aforementioned function.

The claim now follows because

$$\begin{aligned} (p_1 \odot p_2) \odot p_3 &= \left(\bigsqcup_n \alpha_n(\beta_n(p_1) \odot^{(n)} \beta_n(p_2))\right) \odot p_3 \\ &= \bigsqcup_m \alpha_m \left(\beta_m \left(\bigsqcup_n \alpha_n \left(\beta_n(p_1) \odot^{(n)} \beta_n(p_2)\right)\right) \odot^{(m)} \beta_m(p_3)\right) \\ &= \bigsqcup_m \alpha_m \left(\bigsqcup_{n>m} j_{nm} \left(\beta_n(p_1) \odot^{(n)} \beta_n(p_2)\right) \odot^{(m)} \beta_m(p_3)\right) \\ &= \bigsqcup_m \alpha_m \left(\left[\beta_m(p_1) \odot^{(m)} \beta_m(p_2)\right] \odot^{(m)} \beta_m(p_3)\right) \\ &= \bigsqcup_m \alpha_m \left(\beta_m(p_1) \odot^{(m)} \left[\beta_m(p_2) \odot^{(m)} \beta_m(p_3)\right]\right) \\ &= p_1 \odot (p_2 \odot p_3) \end{aligned}$$

The second equality is immediate. □

5.2 Choice

In order to define the other operators it is convenient to have another function Δ available. This function $\Delta : \mathbf{P} \to \mathbf{P}$ removes δ's from the first level of its argument, unless that argument equals $\{\!|\delta|\!\}$. Note that this function is of a 'global' nature, that is, Δ is not linear. It is this definition that enables us to formulate our model. We can define $\Delta_n : P_n \to P_n$ uniformly on the finite elements of P_n by

$$\Delta_n(X) = \begin{cases} X & \text{if } X \equiv \{\langle 0, \delta\rangle\} \\ X \setminus \{\langle 0, \delta\rangle\} & \text{otherwise} \end{cases}$$

It is easy to see that Δ_n is monotonic on the finite elements of P_n and hence extends to a continuous function on P_n. (Note that Δ is *not* monotonic with respect to the Smyth or the Hoare order: consider the sets $\{\langle 0, \delta\rangle, \langle 1, a\rangle\} \sqsubseteq_S \{\langle 0, \delta\rangle\}$.) It is also easy to show that $\Delta_n \circ j_n = j_n \circ \Delta_{n+1}$, hence $[\Delta_n]_n$ is a compatible family. Let $\Delta : \mathbf{P} \to \mathbf{P}$ be the induced function on $\mathbf{P}$. The following lemmas list some elementary properties of Δ.

Lemma 5.4 *For all $p, p' \in \mathbf{P}$,*

1. $\Delta(\Delta(p)) = \Delta(p)$*;*
2. $\Delta(p \uplus p') = \Delta(\Delta(p) \uplus \Delta(p')) = \Delta(p \uplus \Delta(p'))$*;*
3. $\Delta(p \odot p') = \Delta(p) \odot p'$*.*

We define the semantic choice operator $\oplus : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$ as $\oplus = \Delta \circ \uplus$.

Remark. Note that with this definition of the choice operator, it is in general *not* the case that $p \oplus p = p$ or that $p \oplus \{\!|\delta|\!\} = p$. For instance, let $p \equiv \{\langle 0, \delta\rangle, \langle 1, a\rangle\}$. Then

$$p \oplus \{\!|\delta|\!\} = \Delta(\{\langle 0, \delta\rangle, \langle 1, a\rangle\}) = \{\langle 1, a\rangle\} \neq p$$

We call a function f *$\oplus$-linear* if $f(X \oplus Y) = f(X) \oplus f(Y)$.

Lemma 5.5 *If f is linear and $f(\{\!|\delta|\!\}) = \{\!|\delta|\!\}$, then f is $\oplus$-linear.*

Proof Immediate from the observation that $\Delta \circ f(X) = \Delta \circ f(\Delta(X))$. $\square$

Lemma 5.6 *For all $p_1, p_2, p_3 \in \mathbf{P}$,*

1. $p_1 \oplus p_2 = p_2 \oplus p_1$ *(Axiom A1);*
2. $p_1 \oplus (p_1 \oplus p_3) = (p_1 \oplus p_2) \oplus p_3$ *(Axiom A2);*
3. $(p_1 \oplus p_2) \odot p_3 = (p_1 \odot p_3) \oplus (p_2 \odot p_3)$ *(Axiom A4).*

Proof The first equality is trivial. For the other two equalities, we have

$$\begin{aligned} p_1 \oplus (p_2 \oplus p_3) &= \Delta(p_1 \uplus \Delta(p_2 \uplus p_3)) \\ &= \Delta(p_1 \uplus p_2 \uplus p_3) \\ &= \Delta(\Delta(p_1 \uplus p_2) \uplus p_3) \end{aligned}$$

$$
\begin{aligned}
&= (p_1 \oplus p_2) \oplus p_3 \\
(p_1 \oplus p_2) \odot p_3 &= \Delta(p_1 \uplus p_2) \odot p_3 \\
&= \Delta((p_1 \uplus p_2) \odot p_3) \\
&\stackrel{(*)}{=} \Delta(p_1 \odot p_3 \uplus p_2 \odot p_3) \\
&= (p_1 \odot p_3) \oplus (p_2 \odot p_3)
\end{aligned}
$$

where equality $(*)$ holds since $\odot$ is defined pointwise in its first argument and hence is linear in that argument. □

5.3 Parallel composition

We now define the parallel composition operator $\otimes : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$ as follows. We now define the parallel composition operator $\otimes : \mathbf{P} \times \mathbf{P} \to \mathbf{P}$. First we discuss how this function should behave. Let p_1 and p_2 be two elements in $\mathbf{P}$. Then, according to Axiom $CM1$, $p_1 \otimes p_2$ should equal $(p_1 \otimes_L p_2) \oplus (p_2 \otimes_L p_1) \oplus (p_1 \oplus\!\!\!| \; p_2)$ where $\otimes_L$ denotes the semantic left-merge and $\oplus\!\!\!|$ denotes the semantic communication merge function, respectively. Since $\otimes_L$ should execute an action from its left-hand-side argument first, and then behave like $\otimes$, its definition closely resembles the definition of the sequential composition function $\odot$, except that in the recursion we have to apply $\otimes$ instead of $\odot$. Hence the higher-order operator Ψ used in the definition of sequential composition in section 5.1, will again be used in the definition of the left-merge function.

For communications, we have to take an action from the left-hand-side and one form the right-hand-side argument, and attempt to synchronize them. To do this, we lift the function γ as given in the definition of the algebra to a function

$$\gamma : (\{\delta\}_\perp + A_\perp) \times (\{\delta\}_\perp + A_\perp) \to (\{\delta\}_\perp + A_\perp)$$

in the obvious strict way. Hence γ is strict, commutative, and associative and has $\langle 0, \delta \rangle$ as zero. We now define

$$
\begin{aligned}
\Psi_C \quad : \quad & [\mathbf{P} \times \mathbf{P} \to \mathbf{P}] \to \\
& [(\{\delta\}_\perp + A_\perp + A_\perp \times \mathbf{P}) \times (\{\delta\}_\perp + A_\perp + A_\perp \times \mathbf{P}) \to (\{\delta\}_\perp + A_\perp + A_\perp \times \mathbf{P})]
\end{aligned}
$$

by

$$
\begin{array}{rl}
\Psi_C(f)(x,y) = & \textbf{if } (is_0(x) \vee is_1(x)) \wedge (is_0(y) \vee is_1(y)) \\
& \textbf{then } \gamma(x,y) \\
& \textbf{else if } is_1(x) \wedge is_2(y) \\
& \qquad \textbf{then } \; \textbf{if } is_0(\gamma(x,\pi(y))) \\
& \qquad \qquad \textbf{then } \delta \\
& \qquad \qquad \textbf{else } \langle \gamma(x,\pi(y)), \pi'(y) \rangle \\
& \qquad \textbf{else } \; \textbf{if } is_2(x) \wedge is_1(y) \\
& \qquad \qquad \textbf{then } \; \textbf{if } is_0(\gamma(\pi(x),y)) \\
& \qquad \qquad \qquad \textbf{then } \delta \\
& \qquad \qquad \qquad \textbf{else } \langle \gamma(\pi(x),y), \pi'(x) \rangle \\
& \qquad \qquad \textbf{else if } is_0(\gamma(\pi(x),\pi(y))) \\
& \qquad \qquad \qquad \textbf{then } \delta \\
& \qquad \qquad \qquad \textbf{else } \langle \gamma(\pi(x),\pi(y)), f(\pi'(x),\pi'(y)) \rangle
\end{array}
$$

and we define $\tilde{\Psi}_C = \lambda f.\Delta \circ (\Psi_C(f))^{\ddagger}$.

We now define

$$\Phi(f)(p_1,p_2) = \tilde{\Psi}(f)(p_1,p_2) \oplus \tilde{\Psi}(f)(p_2,p_1) \oplus \tilde{\Psi}_C(f)(p_1,p_2)$$

and put $\otimes = fix(\Phi)$. Likewise, we define $\otimes_L = \tilde{\Psi}(\otimes)$ and $\mid\; = \tilde{\Psi}_C(\otimes)$. The next lemma follows immediately from the definitions.

Lemma 5.7 *For all $a, b, c \in A$, $p, p' \in \mathbf{P}$,*

1. $\{\!|a|\!\} \mid \{\!|b|\!\} = \{\!|b|\!\} \mid \{\!|a|\!\} = \{\!|\gamma(a,b)|\!\}$ *(Axiom CF);*
2. $(\{\!|a|\!\} \mid \{\!|b|\!\}) \mid \{\!|c|\!\} = \{\!|a|\!\} \mid (\{\!|b|\!\} \mid \{\!|c|\!\}) = \{\!|\gamma(\gamma(a,b),c)|\!\}$*;*
3. $\{\!|a|\!\} \otimes_L p = \{\!|\langle a,p\rangle|\!\} = \{\!|a|\!\} \odot p$ *(Axiom $CM2$);*
4. $(\{\!|a|\!\} \odot p) \otimes_L p' = \{\!|a|\!\} \odot (p \otimes p')$ *(Axiom $CM3$);*
5. $(\{\!|a|\!\} \odot p) \mid \{\!|b|\!\} = \{\!|\gamma(a,b)|\!\} \odot p$ *(Axiom $CM5$);*
6. $\{\!|a|\!\} \mid (\{\!|b|\!\} \odot p) = \{\!|\gamma(a,b)|\!\} \odot p$ *(Axiom $CM6$);*
7. $(\{\!|a|\!\} \odot p) \mid (\{\!|b|\!\} \odot p') = \{\!|\gamma(a,b)|\!\} \odot (p \otimes p')$ *(Axiom $CM7$).*

Lemma 5.8 *For all $p_1, p_2, p_3 \in \mathbf{P}$,*

1. $p_1 \otimes p_2 = (p_1 \otimes_L p_2) \oplus (p_2 \otimes_L p_1) \oplus (p_1 \mid p_2)$ *(Axiom $CM1$);*
2. $(p_1 \oplus p_2) \otimes_L p_3 = (p_1 \otimes_L p_3) \oplus (p_2 \otimes_L p_3)$ *(Axiom $CM4$);*
3. $(p_1 \oplus p_2) \mid p_3 = (p_1 \mid p_3) \oplus (p_2 \mid p_3)$ *(Axiom $CM8$);*
4. $p_1 \mid (p_2 \oplus p_3) = (p_1 \mid p_2) \oplus (p_1 \mid p_3)$ *(Axiom $CM9$).*

Proof The first equality follows immediately from the definitions. For the second equality, we have $\Delta(p \otimes_L p') = \Delta(p) \otimes_L p'$ just like for $\odot$. The claim now follows by the same argument. The third equality follows from Lemma 5.5. The fourth equality is proved similarly. $\square$

Having defined the merge, left-merge and communication merge functions, we now show that the Standard Concurrency Axioms hold for this interpretation. First, we have that $\otimes^{(n)}$ is given by:

$$\otimes^{(n)}(p, p') = \otimes_L^{(n)}(p, p') \oplus^{(n)} \otimes_L^{(n)}(p', p) \oplus^{(n)} \obar^{(n)}(p, p')$$

where $\obar^{(n)}$ is inductively given by

$$\begin{array}{rcl} \tilde{\obar}^{(0)} & = & \lambda x, y.\ \bot \\ \tilde{\obar}^{(n+1)} & = & \Psi_C\left(\obar^{(n)}\right) \end{array}$$

putting $\obar^{(n)} = \Delta \circ \left(\tilde{\obar}^{(n)}\right)^{\ddagger}$. Furthermore, $\otimes_L^{(n)}$ is given analogously to $\odot^{(n)}$.

$$\begin{array}{rcll} \tilde{\otimes}_L^{(0)} & = & \lambda x, y. & \bot \\ \tilde{\otimes}_L^{(n+1)} & = & \lambda x, y. & \textbf{if } is_0(x) \textbf{ then } x \\ & & & \textbf{else if } is_1(x) \textbf{ then } \langle x, j_n(y)\rangle \\ & & & \qquad \textbf{else } \langle \pi(x), \pi'(x) \otimes^{(n)} j_n(y)\rangle \end{array}$$

Lemma 5.9 *For all $p, p' \in \mathbf{P}$,*

1. $p \obar p' = p' \obar p$ *(Axiom SC3);*
2. $p \otimes p' = p' \otimes p$ *(Axiom SC4).*

Proof We can prove that, for all $p, p' \in P_n$, $p \obar^{(n)} p' = p' \obar^{(n)} p$ and $p \otimes^{(n)} p' = p' \otimes^{(n)} p$ by a simultaneous induction on n. The first equality follows easily from the fact that $\tilde{\obar}^{(n)}$ is defined symmetrically in x and y, and the induction hypothesis. The claim for $\otimes^{(n)}$ is then trivial. $\square$

Lemma 5.10 *For all $p_1, p_2, p_3 \in \mathbf{P}$,*

1. $(p_1 \otimes_L p_2) \otimes_L p_3) = p_1 \otimes_L (p_2 \otimes p_3)$ *(Axiom SC1);*
2. $(p_1 \obar p_2) \otimes_L p_3 = p_1 \obar (p_2 \otimes_L p_3)$ *(Axiom SC2);*
3. $p_1 \obar (p_2 \obar p_3) = (p_1 \obar p_2) \obar p_3$ *(Axiom SC5);*
4. $p_1 \otimes (p_2 \otimes p_3) = (p_1 \otimes p_2) \otimes p_3$ *(Axiom SC6).*

Proof We prove the equalities in their $(\cdot)^{(n)}$ form, by simultaneous induction on n. The base case $n = 0$ is in all cases trivial. For the first equality, we have

$$\begin{array}{rcl} \tilde{\otimes}_L^{(n+1)}(x, \otimes^{(n+1)}(p_2, p_3)) & = & \textbf{if } is_0(x) \textbf{ then } x \\ & & \textbf{else } \ \textbf{if } is_1(x) \\ & & \qquad \textbf{then } \langle x, j_n(\otimes^{(n+1)}(p_2, p_3))\rangle \\ & & \qquad \textbf{else } \langle \pi(x), \otimes^{(n)}(\pi'(x), j_n(\otimes^{(n+1)}(p_2, p_3)))\rangle \end{array}$$

$$\begin{array}{rcl}\tilde{\otimes}_L^{(n+1)}(\tilde{\otimes}_L^{(n+1)}(x,p_2),p_3) & = & \textbf{if } is_0(x) \textbf{ then } x \\ & & \textbf{else } \textbf{if } is_1(x) \\ & & \qquad \textbf{then } \langle x, \otimes^{(n)}(j_n(p_2), j_n(p_3))\rangle \\ & & \qquad \textbf{else } \langle \pi(x), \otimes^{(n)}(\otimes^{(n)}(\pi'(x), j_n(p_2)), j_n(p_3))\rangle \end{array}$$

By properties of the compatible family $[\otimes^{(n)}]_n$ and the induction hypothesis on $\otimes^{(n)}$ these two expressions are the same.

The second equality is proved likewise.

The associativity of $\between^{(n+1)}$ follows readily from the associativity of γ and the induction hypothesis on $\otimes^{(n)}$.

For the last equality, we write out left- and right-hand-sides of the equality.

$$\begin{array}{l} p_1 \otimes^{(n+1)} (p_2 \otimes^{(n+1)} p_3) = \\ \quad [p_1 \otimes_L^{(n+1)} (p_2 \otimes^{(n+1)} p_3)]^{\{1\}} \oplus^{(n+1)} [(p_2 \otimes_L^{(n+1)} p_3) \otimes_L^{(n+1)} p_1]^{\{2\}} \oplus^{(n+1)} \\ \quad [(p_3 \otimes_L^{(n+1)} p_2) \otimes_L^{(n+1)} p_1]^{\{3\}} \oplus^{(n+1)} [(p_2 \between^{(n+1)} p_3) \otimes_L^{(n+1)} p_1]^{\{4\}} \oplus^{(n+1)} \\ \quad [p_1 \between^{(n+1)} (p_2 \otimes_L^{(n+1)} p_3)]^{\{5\}} \oplus^{(n+1)} [p_1 \between^{(n+1)} (p_3 \otimes_L^{(n+1)} p_2)]^{\{6\}} \oplus^{(n+1)} \\ \quad [p_1 \between^{(n+1)} (p_2 \between^{(n+1)} p_3)]^{\{7\}}. \end{array}$$

$$\begin{array}{l} (p_1 \otimes^{(n+1)} p_2) \otimes^{(n+1)} p_3 = \\ \quad [(p_1 \otimes_L^{(n+1)} p_2) \otimes_L^{(n+1)} p_3]^{\{1'\}} \oplus^{(n+1)} [(p_2 \otimes_L^{(n+1)} p_1) \otimes_L^{(n+1)} p_3]^{\{2'\}} \oplus^{(n+1)} \\ \quad [(p_1 \between^{(n+1)} p_2) \otimes_L^{(n+1)} p_3]^{\{5'\}} \oplus^{(n+1)} [p_3 \otimes_L^{(n+1)} (p_1 \otimes^{(n+1)} p_2)]^{\{3'\}} \oplus^{(n+1)} \\ \quad [(p_1 \otimes_L^{(n+1)} p_2) \between^{(n+1)} p_3]^{\{6'\}} \oplus^{(n+1)} [(p_2 \otimes_L^{(n+1)} p_1) \between^{(n+1)} p_3]^{\{4'\}} \oplus^{(n+1)} \\ \quad [(p_1 \between^{(n+1)} p_2) \between^{(n+1)} p_3]^{\{7'\}}. \end{array}$$

In the above we have, by the previous cases in the lemma, that $\{i\} = \{i'\}$ for $1 \leq i \leq 7$. $\square$

5.4 Encapsulation

Each subset $H \subseteq A$ gives rise to the following continuous function $\widehat{H} : A_\perp \to \mathbf{T}$:

$$\widehat{H}(x) = \begin{cases} \perp & \text{if } x \equiv \perp \\ \mathbf{t} & \text{if } x \not\equiv \perp \text{ and } x \in H \\ \mathbf{f} & \text{otherwise} \end{cases}$$

Fixing such a set $H \subseteq A$, we define $\nabla_H : \mathbf{P} \to \mathbf{P}$ with the help of

$$\Psi_H : (\mathbf{P} \to \mathbf{P}) \to (\{\delta\}_\perp + A_\perp + A_\perp \times \mathbf{P}) \to (\{\delta\}_\perp + A_\perp + A_\perp \times \mathbf{P})$$

by

$$\begin{array}{rl}\Psi_H(f)(x) = & \textbf{if } is_1(x) \\ & \textbf{then if } \widehat{H}(x) \textbf{ then } \delta \textbf{ else } x \\ & \textbf{else if } is_2(x) \\ & \qquad \textbf{then if } \widehat{H}(\pi(x)) \textbf{ then } \delta \textbf{ else } \langle \pi(x), f(\pi'(x))\rangle \\ & \qquad \textbf{else } x \end{array}$$

Now define $\tilde{\Psi}_H = \lambda f.\Delta \circ \phi^{-1} \circ \mathcal{P}^*(\Psi_H(f)) \circ \phi$ and $\nabla_H = fix(\tilde{\Psi}_H)$. We give some elementary properties of ∇_H.

Lemma 5.11 *Let $H \subseteq A$ and let $a \in A$. Then*

1. $\nabla_H(\{\!|a|\!\}) = \{\!|\delta|\!\}$ *if $a \in H$ (Axiom D1);*
2. $\nabla_H(\{\!|a|\!\}) = \{\!|a|\!\}$ *if $a \notin H$ (Axiom D2).*

Lemma 5.12 *For all $p, p' \in \mathbf{P}$,*

1. $\nabla_H(p) = \nabla_H(\Delta(p)) = \Delta(\nabla_H(p))$;
2. $\nabla_H(p \uplus p') = \Delta(\nabla_H(p) \uplus \nabla_H(p'))$;
3. $\nabla_H(p \oplus p') = \nabla_H(p) \oplus \nabla_H(p')$ *(Axiom D3).*

Proof We have

$$\begin{array}{rcl}\nabla_H(p \oplus p') & = & \nabla_H(\Delta(p \uplus p')) \\ & = & \Delta(\nabla_H(p) \uplus \nabla_H(p')) \\ & = & \nabla_H(p) \oplus \nabla_H(p')\end{array}$$

The other equalities follow immediately from the definitions. □

Lemma 5.13 *For all $p_1, p_2 \in \mathbf{P}$, $\nabla_H(p_1 \odot p_2) = \nabla_H(p_1) \odot \nabla_H(p_2)$ (Axiom D4).*

Proof Again define the compatible family $\{\nabla_H^{(n)} : P_n \to P_n : n < \omega\}$ and use induction on n to prove that for each n,

$$\nabla_H^{(n)}(p \odot^{(n)} p') = \nabla_H^{(n)}(p) \odot^{(n)} \nabla_H^{(n)}(p')$$

for all $p, p' \in P_n$. □

5.5 The denotational model

Having defined semantical counterparts to all syntactic operators, we are ready to define the denotational semantics. This definition is the standard one, compare [dB80]. First of all, let $\Gamma : PVar \to \mathbf{P}$ be the set of *environments* or meanings of procedure variables.

Definition 5.14 *We define $D : \mathcal{L} \to \Gamma \to \mathbf{P}$ by induction on the structure of s as follows:*

- $D(\gamma)(a) = \{\!|a|\!\}$;
- $D(\gamma)(\delta) = \{\!|\delta|\!\}$;

- $D(\gamma)(s_1; s_2) = D(\gamma)(s_1) \odot D(\gamma)(s_2)$;
- $D(\gamma)(s_1 + s_2) = D(\gamma)(s_1) \oplus D(\gamma)(s_2)$;
- $D(\gamma)(s_1 \lfloor\!\lfloor s_2) = D(\gamma)(s_1) \otimes_L D(\gamma)(s_2)$;
- $D(\gamma)(s_1 | s_2) = D(\gamma)(s_1) \ \oplus\!\!\!|\ D(\gamma)(s_2)$;
- $D(\gamma)(s_1 \parallel s_2) = D(\gamma)(s_1) \otimes D(\gamma)(s_2)$;
- $D(\gamma)(\partial_H(s)) = \nabla_H(D(\gamma)(s))$;
- $D(\gamma)(X) = \gamma(X)$.

The order on $\mathbf{P}$ extends to an order on Γ. The higher-order operator $\Upsilon : \Gamma \to \Gamma$, defined by $\Upsilon(\gamma)(X) = D(\gamma)(d(X))$, is a continuous operator and hence has a fixed point γ_d. Now define $[\![-]\!] : \mathcal{L} \to \mathbf{P}$ as $D(\gamma_d)$.

6 Soundness and completeness

In the preceding section we have defined a denotational semantics for the language $\mathcal{L}$. In this section we show that this interpretation is sound and complete for the process algebra *ACP*.

In view of the lemmas in the preceding section the only equalities that we still need to prove are Axioms $A3$ and $A6$. We can define the following subset $\mathbf{P}^* \subset \mathbf{P}$. First, define the operator $\Delta^* : \mathbf{P} \to \mathbf{P}$ as

$$\begin{aligned}\Theta(f)(x) = \ & \textbf{if } is_0(x) \vee is_1(x) \textbf{ then } x \\ & \textbf{else } \langle \pi(x), f(\pi'(x)) \rangle\end{aligned}$$

and set $\tilde{\Theta} = \lambda f.\Delta \circ \phi^{-1} \circ \mathcal{P}^*(\Theta(f)) \circ \phi$. Set $\Delta^* = fix(\tilde{\Theta})$. Intuitively, Δ^* applies Δ recursively to a process. Define $\mathbf{P}^* = \Delta^*(\mathbf{P})$, the direct image of $\mathbf{P}$ under Δ^*. As $\Delta^*(\Delta^*(p)) = \Delta^*(p)$, $\mathbf{P}^*$ consists of all fixed points of Δ^*. For the next proposition we need the following definition. Given a domain D, a subset $D' \subseteq D$ is called *inclusive* if whenever $(x_i)_i \subseteq D'$ then $\bigsqcup x_i \in D'$.

Proposition 6.1 $\mathbf{P}^*$ *is an inclusive subset of* $\mathbf{P}$.

Since $\perp$, $[\![a]\!]$ and $[\![\delta]\!] \in \mathbf{P}^*$ and all operators preserve the property of being in $\mathbf{P}^*$, it follows that $[\![-]\!]$ is a function from $\mathcal{L}$ to $\mathbf{P}^*$.

Lemma 6.2 *For all* $p \in \mathbf{P}^*$,

1. $p \oplus p = p$ *(Axiom* $A3$*)*;
2. $p \oplus \{\!|\delta|\!\} = p$ *(Axiom* $A6$*)*.

Finally, we have to show that the Approximation Induction Principle holds in the model given by the denotational semantics. We define projections $\pi_n : \mathbf{P} \to P_n$ as follows. Define $\tilde{\pi}_n$ by

$$\begin{array}{rcl}\tilde{\pi}_1(x) & = & \textbf{if } is_0(x) \vee is_1(x) \textbf{ then } x \\ & & \textbf{else } \pi(x) \\ \tilde{\pi}_{n+1}(x) & = & \textbf{if } is_0(x) \vee is_1(x) \textbf{ then } x \\ & & \textbf{else } \langle \pi(x), \pi_n(\pi'(x)) \rangle \end{array}$$

and set $\pi_n = \mathcal{P}^*(\tilde{\pi}_n)$.

We have the following lemma.

Lemma 6.3 *For all $s \in \mathcal{L}$, $[\![\pi_n(s)]\!] = \pi_n[\![s]\!]$.*

Proof The lemma obviously holds for $s \in \mathcal{N}$. For $s \in \mathcal{L}_\mathrm{f}$, we have $[\![s]\!] = [\![\mathcal{NF}(s)]\!]$. Hence $[\![\pi_n(s)]\!] = [\![\pi_n(\mathcal{NF}(s))]\!] = \pi_n[\![\mathcal{NF}(s)]\!] = \pi_n[\![s]\!]$. For $s \in \mathcal{L}$ we first observe $\vdash \pi_n(s) = \pi_n(s')$ where s' is obtained from $s^{(n)}$ (see section 2) by replacing each procedure variable by an arbitrary term in $\mathcal{L}_\mathrm{f}$. This follows easily from the fact that we have guarded recursion. Likewise, $\pi_n[\![s]\!] = \pi_n[\![s']\!]$. Hence $[\![\pi_n(s)]\!] = [\![\pi_n(s')]\!] = \pi_n[\![s']\!] = \pi_n[\![s]\!]$. □

We now discuss the relation between π_n and β_n. Recall the description of β_n as given in the proof of Lemma 5.1. Comparing this last definition with the definition of π_n above, we see that the only difference between these two functions is that β_n maps a set on depth $n+1$ to $\perp$ whereas π_n removes the set alltogether. Hence we have the following proposition.

Lemma 6.4 *For all n, $p, p' \in \mathbf{P}$,*

1. *$\beta_n(p) = \beta_n(p')$ implies $\pi_n(p) = \pi_n(p')$;*
2. *$\pi_{n+1}(p) = \pi_{n+1}(p')$ implies $\beta_n(p) = \beta_n(p')$.*

Proposition 6.5 (AIP) *For all $p, p' \in \mathbf{P}$, $p = p'$ iff for all n, $\pi_n(p) = \pi_n(p')$.*

Proof It folows from section 4 that for all $p, p' \in \mathbf{P}$, $p = p'$ iff for all n it is the case that $\beta_n(p) = \beta_n(p')$. The claim now follows from Lemma 6.4. □

Now we have shown that every axiom from Table 1 and the rule AIP hold in the model.

To obtain an interpretation of the collection of all terms $\mathcal{T}$, we need to introduce environments. An environment is a function $\rho : Var \to \mathbf{P}^*$. Given an environment ρ, the interpretation $[\![-]\!] : \mathcal{L} \to \mathbf{P}^*$ extends to an interpretation $[\![-]\!]_\rho : \mathcal{T} \to \mathbf{P}^*$ by stipulating that $[\![x]\!]_\rho = \rho(x)$. Note that, for every closed term s and environment ρ, $[\![s]\!] = [\![s]\!]_\rho$. We have the following theorem.

Theorem 6.6 (Soundness) *For all $s, t \in \mathcal{T}$, $\vdash s = t$ implies $[\![s]\!]_\rho = [\![t]\!]_\rho$ for every environment ρ.*

We now show that the semantics is *complete* in the sense that the reverse implication also holds, at least for closed terms. We obtain this result rather immediately from the fact that $[\![-]\!]$ is a sound model in which two distinct normal forms have different interpretations.

First, recall the collection of normal forms of terms as given in section 2. We can assign to each total $p \in \Delta^*(P_n)$ an element $\mathcal{NF}(p) \in \mathcal{N}$, where we call p *total* if it has no occurrences of $\perp$. We proceed by an induction on n.

$$(n=1)\ \ \mathcal{NF}(p) = \begin{cases} \delta & p \equiv \{\langle 0, \delta\rangle\} \\ \sum_{i=1}^{n} a_i & p \equiv \{\langle 1, a_1\rangle, \ldots, \langle 1, a_n\rangle\} \end{cases}$$

$$(n+1)\ \ \mathcal{NF}(p) = \begin{cases} \delta & p \equiv \{\langle 0, \delta\rangle\} \\ \sum_{i=1}^{n} a_i + \sum_{k=1}^{m} a'_k ; \mathcal{NF}(p_k) & p \equiv \left\{ \begin{array}{c} \langle 1, a_1\rangle, \ldots, \langle 1, a_n\rangle, \\ \langle 2, \langle a'_1, p_1\rangle\rangle, \ldots, \langle 2, \langle a'_m, p_m\rangle\rangle \end{array} \right\} \end{cases}$$

Note that we have to "choose" an order in which to list the elements of p in the definition of $\mathcal{NF}$. In view of Axioms $A1$ and $A2$, however, this order is irrelevant.

The following proposition shows that the functions $\mathcal{NF}$ and $[\![-]\!]$ are inverse to each other.

Lemma 6.7 *1. For all total $p \in \bigcup_{n<\omega} \Delta^*(P_n)$, $p = [\![\mathcal{NF}(p)]\!]$.*

2. For all $s \in \mathcal{N}$, $\vdash s = \mathcal{NF}([\![s]\!])$.

Corollary 6.8 *For all $s, t \in \mathcal{N}$, $\vdash s = t$ if and only if $[\![s]\!] = [\![t]\!]$.*

Using this corollary, we can show that the model $[\![-]\!]$ is complete for the equational theory of the finite terms $\mathcal{L}_{\mathrm{f}}$.

Proposition 6.9 *For all $s, t \in \mathcal{L}_f$, $\vdash s = t$ iff $[\![s]\!] = [\![t]\!]$.*

Proof We have

$$\begin{array}{rl} \vdash s = t & \text{iff}\ \ \vdash \mathcal{NF}(s) = \mathcal{NF}(t) \\ & \text{iff}\ \ [\![\mathcal{NF}(s)]\!] = [\![\mathcal{NF}(t)]\!] \\ & \text{iff}\ \ [\![s]\!] = [\![t]\!] \end{array}$$

where the latter equivalence holds since, by soundness, we have that $[\![s]\!] = [\![\mathcal{NF}(s)]\!]$. □

The previous proposition extends immediately to the whole of $\mathcal{L}$.

Theorem 6.10 *For all $s, t \in \mathcal{L}$, $\vdash s = t$ if and only if $[\![s]\!] = [\![t]\!]$.*

Proof We have

$$\begin{array}{rll} \vdash s = t & \text{iff} & \forall n. \vdash \pi_n(s) = \pi_n(t) \\ & \text{iff} & \forall n. [\![\pi_n(s)]\!] = [\![\pi_n(t)]\!] \\ & \text{iff} & \forall n. \pi_n [\![s]\!] = \pi_n [\![t]\!] \\ & \text{iff} & [\![s]\!] = [\![t]\!]. \end{array}$$

□

Since the equational theory precisely captures bisimulation, the following corollary is immediate.

Corollary 6.11 *For all $s, t \in \mathcal{L}$, s and t are bisimular if and only if $[\![s]\!] = [\![t]\!]$.*

7 Discussion

We have defined an order theoretic interpretation for $\mathcal{L}$ and have proved that it is a complete model for the equational theory of *ACP*. Some remarks seem appropriate. First of all, traditionally *ACP* is interpreted over so-called process graphs modulo strong bisimulation [BW91]. A canonical model for the algebra of finite terms (*i.e.* without procedure variables) is the set of finitely branching trees of finite depth. These trees precisely correspond to finite an toatal (*i.e.* not containing $\perp$) elements in $\mathbf{P}^*$. Next, a recursively specified process X can be given meaning in the *projective limit model* [BW91]. This is essentially similar to how the denotation of X is obtained in $\mathbf{P}$.

Next, $\mathbf{P}$ contains a lot of 'junk', *i.e.* elements $p \notin \mathbf{P}^*$. Moreover, $\mathbf{P}^*$ itself contains junk: no non-total process (that is, a process containing $\perp$) can be obtained as the denotation of a closed term. Moreover, consider the infinite process $\{\langle 1, a\rangle : a \in A\} \cup \{\perp\}$. Since the recursion is guarded, this process can never be obtained as the denotation of a program.

An important feature of process algebras is the so-called *silent move* τ, which is used to model a step or a sequence of steps local to a process [BW91, dBK90, Mil89]. Axioms for τ include $\tau; x + x = \tau; x$. It is readily seen that this axiom prevents us from modeling the semantic choice operator by a continuous function. Hence we cannot extend our model to cover this extension of the algebra. It is an interesting question which kind of structure we should use in order to model this.

References

[Abr91a] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.

[Abr91b] S. Abramsky. Domain theory in logical form. *Ann. Pure and Applied Logic*, 51:1–77, 1991.

[ABV92] L. Aceto, B. Bloom, and F. Vaandrager. Turning SOS rules into equations. In *Proc. 7th Ann. Symp. on Logic in Computer Science*, pages 113–124. IEEE Press, 1992. Full version available as CWI Report CS-R9218.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[BK85] J.A. Bergstra and J.W. Klop. Algebra for communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[BW91] J. Baeten and P. Weyland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge Univ. Press, Cambridge, 1991.

[dB80] J.W. de Bakker. *Mathematical Theory of Program Correctness*. Prentice-Hall International, Englewood-Cliffs, NJ, 1980.

[dBK90] J.W. de Bakker and J.N. Kok. Comparative metric semantics for Concurrent Prolog. *Theoretical Computer Science*, 75:15–43, 1990.

[dBMOZ88] J.W. de Bakker, J.-J.Ch. Meyer, E.-R Olderog, and J.I. Zucker. Transition systems, metric spaces and ready sets in the semantics of uniform concurrency. *Journal of Computer and System Sciences*, 36(2):158–224, 1988.

[dBR92] J.W. de Bakker and J.J.M.M. Rutten, editors. *Ten Years of Concurrency Semantics*. World Scientific, Singapore, 1992.

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

[GS90] C.A. Gunter and D.S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, chapter 12, pages 635–675. Elsevier, Amsterdam, 1990.

[GV92] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–261, 1992.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, Englewood Cliffs, NJ, 1985.

[Hor89] E. Horita. Homomorphism from LTS's to de Bakker-Zucker domain. Unpublished manuscript, 1989.

[HP79] M. Hennessy and G.D. Plotkin. Full abstraction for a simple parallel programming language. In J. Bečvář, editor, *Proc. 8th*

Mathematical Foundations of Computer Science, volume 74 of *Lecture Notes in Computer Science*, pages 108–120. Springer Verlag, Berlin, 1979.

[Kni93a] P.M.W. Knijnenburg. Algebraic domains, chain completion and the Plotkin powerdomain construction. Technical Report RUU-CS-93-03, Utrecht University, 1993.

[Kni93b] P.M.W. Knijnenburg. Guarded structured operational semantics induce complete denotational models. Submitted to: Mathematical Structures in Computer Science, 1993.

[Kni93c] P.M.W. Knijnenburg. *Order-theoretic and Categorical Approaches to Programming Language Semantics*. PhD thesis, Dept. of Computer Science, Utrecht University, 1993.

[LS86] J. Lambek and P.J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge Studies in Advanced Mathematics. Cambridge Univ. Press, Cambridge, 1986.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, NJ, 1989.

[MM79] G. Milne and R. Milner. Concurrent processes and their syntax. *JACM*, 26(2):302–321, 1979.

[Mol90] R. Moller. The importance of the left merge operator. In M. Paterson, editor, *Proceedings* 17^{th} *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 752–764. Springer Verlag, Berlin, 1990.

[Plo79] G.D. Plotkin. A powerdomain construction. *SIAM J. on Computing*, 5:452–487, 1979.

[Plo81a] G.D. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the Pisa lecture notes). Technical report, Dept. of Computer Science, Univ. of Edinburgh, 1981.

[Plo81b] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Dept., Aarhus Univ., 1981.

[RT93] J.J.M.M. Rutten and D. Turi. On the foundations of final semantics: Non-standard sets, metric spaces, partial orders. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proc. REX Workshop on Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 1993.

[Rut90] J.J.M.M. Rutten. Deriving denotational models for bisimulation from Structured Operational Semantics. In M. Broy and C.B. Jones, editors, *Programming concepts and methods*, pages 155–177. North-Holland, Amsterdam, 1990.

[Rut92] J.J.M.M. Rutten. Processes as terms: Non-well-founded models for bisimulation. *Mathematical Structures in Computer Science*, 2(3):257–277, 1992.

[Sco76] D.S. Scott. Datatypes as lattices. *SIAM J. on Computing*, 5:522–587, 1976.

[Smy78] M.B. Smyth. Power domains. *Journal of Computer and System Sciences*, 16:23–36, 1978.

[SP82] M.B. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. on Computing*, 11(4):761–783, 1982.

[Vaa89] F.W. Vaandrager. *Algebraic techniques for concurrency and their applications.* PhD thesis, Universiteit van Amsterdam, 1989.

[vD83] D. van Dalen. *Logic and Structure (2nd Edition).* Springer Verlag, Berlin, 1983.