A Semantic Theory for Value–Passing Processes Late Approach

Anna Ingólfsdóttir

BRICS*
Department of Mathematics and Computer Science
Aalborg University, Denmark

1 Introduction

In the original work of Milner, [Mil80], on *CCS* and Hoare, [Hoa78], on *CSP*, processes are allowed to exchange data in communications. In these original calculi the value-passing calculus is interpreted in terms of the pure calculus in which communication is pure synchronization. A prefixing with an input action, $\overline{a}(x).p$, is interpreted as a non-deterministic choice between pure terms of the form $\overline{a}_v.p[v/x]$, where v ranges over the set of possible values, which in many cases is infinite. In this approach, two processes that synchronize are both supposed to know each other's *channel* and *value*, i.e. the data variable is instantiated by the potential input values already when the process reports the willingness or ability to communicate on the channel c.

In a more recent work on the π-calculus, [MPW89], this semantic approach is referred to as *early semantics* due to the early instantiation of the data variables as described above. Its counterpart, the *late semantics*, is also introduced in the same reference. Here the idea is that the processes *only* synchronize on the channel name and that the inputting process has to accept whatever value the outputting process has to offer. This may be interpreted as if the result of the reception of the value is delayed until the process has received the value. The inputting process reports the willingness to communicate on a channel, c, by performing an action of the form $\overline{c}$, and thereby evolves to a function which waits for the value the outputting counterpart in the communication provides. Symmetrically the result of reporting the willingness to output an uninterpreted value on the channel c is modelled by the action c. By performing this action the process evolves to a term which basically consists of a data expression and a process expressions.

In a more recent version of the π-calculus, the Polyadic π-calculus in [Mil91], the outcome of input and output actions are modelled by extending the syntax

*Basic Research in Computer Science, Centre of the Danish National Research Foundation.

with the new constructions *abstractions* and *concretions*. For further motivation for this approach we refer to [MPW89], [Mil91] and [Ing93].

The aim of this paper is to develop a denotational semantics for languages with simple values based on the late approach described above although we use a slightly different notation as we will explain in the next section.

The paper is divided into two main parts: a general theory to handle value-passing in §2, followed by an application to a concrete definition in §3. In the first part we introduce both a general syntax and a general class of mathematical models to model process algebras with values which support the late semantic approach. This is a direct generalization of the standard Σ–terms and Σ-domains that appear in [He88] to model the pure calculus. For this purpose we introduce the general notion of applicative signature (Σ, C) where Σ is a signature and C a set (of channel names) and that of (Σ, C)-terms. This is basically the syntax of the Polyadic π-calculus presented in [Mil91] although we only allow simple values and allow Σ to be any set of operators. We also introduce the general class of applicative (Σ, C)-domains to model the semantics of the (Σ, C)-terms.

Next we apply the general result for algebraic *cpo*s ([He88]): functions which are monotonic on the compact elements of an algebraic *cpo* can be extended to continuous functions on the whole *cpo* in a unique way. This property enables us to turn an algebraic *cpo* into a (Σ, C)-model by defining the operators on the compact elements and making sure that they are monotonic. In this way we may take their unique continuous extension to be their definition on the whole domain.

In §3 a concrete language and its semantics is given. The language is a modification of the original CCS according to the late approach. This is obtained as the (Σ, C)-terms where Σ is instantiated with the standard operators of CCS. Then a concrete denotational model, *Applicative Communication Trees*, is defined, an instantiation of the general class of (Σ, C)-domains. The carrier, an ω-algebraic *cpo*, is defined as a solution to a recursive domain equation, a direct generalization of Abramsky's model for $SCCS$ presented in [Abr92]. Then we apply the described theory to define the operators.

Finally, in 4, we define a proof system to reason about our language and proof it's soundness and completeness with respect to the denotational semantics derived from the model. The ω-algebraicity together with the construction of the operators enables us to reduce the proof of the completeness and soundness to a proof of the same property for a sublanguage which denotes exactly the compact elements of the model.

2 A General Framework for Late Semantics

In this section we will develop a general theoretical framework for describing denotational semantics for value-passing calculi based on the late approach.

2.1 (Σ, C)-Terms

In this subsection we will extend the standard notion of a signature, Σ, and that of Σ-terms used for the pure calculus in order to model processes with value-passing based on the late approach. We do this by introducing the notion of *applicative signature* as a pair, (Σ, C), where Σ is a signature and C is a set (of channel names) and that of (Σ, C)-terms.

We assume some predefined category of expression, Exp, ranged over by e including a countable set of values, Val, ranged over by v, and a set of value variables, Var, ranged over by x. We also assume a predefined category, $BExp$, of boolean expressions, ranged over by be. $BExp$ should at least include equality between the elements of Exp. From that a membership of a finite set can easily be derived. Value expressions are supposed to be equipped with a substitution, denoted by $e[e'/x]$ and an evaluation function $[\![\_]\!] : Exp \times VEnv \longrightarrow Val$ where $VEnv$ is the set of value environments $\sigma : Var \longrightarrow Val$. For closed expression we write $[\![e]\!]$ instead of $[\![e]\!]\sigma$. Further we preassumes an infinite set of process variables, PN, ranged over by P, Q, etc.

The set of (Σ, C)-*terms* is now given as the triplet

$$T_{(\Sigma,C)} = (Proc_{(\Sigma,C)}, Fun_{(\Sigma,C)}, Pair_{(\Sigma,C)})$$

of the sets generated by Σ and C according to the following syntax.

$$\begin{array}{ll} Proc_{(\Sigma,C)}: & p ::= op(\underline{p}) \mid c?.f \mid c!.\pi \mid \tau.p \mid be \rightarrow p, p' \\ Fun_{(\Sigma,C)}: & f ::= [x]p \\ Pairs_{(\Sigma,C)}: & \pi ::= (e, t) \end{array}$$

If the process names in PN are added as primitives to the syntax, we write $T_{(\Sigma,C)}(PN)$, and $T^{rec}_{(\Sigma,C)}(PN)$ if the recursive binding $rec\_.\_$ is also allowed.

We have three kinds of actions, input actions of the form $c?$, $c \in C$, output actions of the form $c!$, $c \in C$ and the silent action τ. We write $C?$ for $\{c? | c \in C\}$ and $C!$ for $\{c! | c \in C\}$. The set $Act = C! \cup C?$ is ranged over by a whereas $Act_\tau = C! \cup C? \cup \{\tau\}$ is ranged over by μ. The action of inputting on channel c is given by $c?$ whereas the action of outputting on that channel is given by $c!$. The abstractions are given by terms of the form $[x]p$, which we call *function terms*, where x is a data variable and p a process term. The input prefixing becomes $c?.[x]p$. The concretions are of the form (e, p), called *pair terms*, where e is a data expression and p a process term and the output prefixing becomes $c!.(e, p)$. We also assume that we have a set of operators, Σ, which at least is supposed to contain the symbol Ω to model the divergent or completely undecided process. Typically Σ contains the standard CCS operators such as NIL, $+$, $\_|\_$, etc. Now the processes are obtained by the input and output prefixing just described, prefixing with the silent action τ and by applying the operators in Σ.

Prefixing by $[x]$ binds the data variable x and the recursion construct is a binding construct for process variables. A value variable, x, is free if it is not in the scope of a prefix, $[x]$, and a process variable P is free if it is not in the scope of a recursion construct, $rec\ P.\_$. We shall mainly be concerned with expressions which contain no free occurrences of value variables. We denote the set of all process terms, functions terms, etc. with no free occurrences of value variables by $CProc_{(\Sigma,C)}$, $CFun_{(\Sigma,C)}$, etc. These will be referred to as processes, functions, etc. ranged over by cp, cf, etc. For $f = [x]p$ we use the convention $f(v) = ([x]p)(v) = p[v/x]$. We assume a notion of substitution for both data and process variables for terms.

2.2 (Σ, C)-Orders and (Σ, C)-Domains

Now we define applicative orders and applicative ordered (Σ, C)-algebras where we use the abbreviations *pro* for pre-order, *po* for partial order and *cpo* for complete partial order:

Definition 2.1 [Applicative Orders] A pair $\langle \overline{A}, \sqsubseteq_A \rangle$ is an applicative *pro/po/cpo* if $\overline{A} = (A_{proc}, A_{fun}, A_{pair})$ is such that

1. $\langle A_{proc}, \sqsubseteq_{A_{proc}} \rangle$ is a *pro/po/cpo*
2. $A_{fun} \subseteq Val \longrightarrow A_{proc}$ and $A_{pair} \subseteq Val \times A_{proc}$ are *pro/po/cpo*s with the standard induced ordering (i.e. $\sqsubseteq_{A_{fun}}$ is the pointwise ordering and $\sqsubseteq_{A_{pair}}$ is defined by: $(v_1, p_1) \sqsubseteq_{A_{pair}} (v_2, p_2)$ if $v_1 = v_2$ and $p_1 \sqsubseteq_{A_{proc}} p_2$).

$\overline{A}$ is said to be *fully applicative* if $A_{proc} = A$, $A_{fun} = Val \longrightarrow A$ and $A_{pair} = Val \times A$ for some A. In that case we refer to $\overline{A}$ as A. An applicative *cpo* is said to be algebraic/ω-algebraic if A_{proc}, A_{fun} and A_{pair} are algebraic/ω-algebraic *cpo*s.

$\square$

Definition 2.2 [(Σ, C)-Orders] A four tuple $\langle \overline{A}, \sqsubseteq_{\overline{A}}, \Sigma_{\overline{A}}, C_{\overline{A}} \rangle$ is an applicative $(\Sigma, C) -$ *pro/po/cpo* if $\overline{A} = (A_{proc}, A_{fun}, A_{pair})$ is such that

1. $\langle \overline{A}, \sqsubseteq_{\overline{A}} \rangle$ is an applicative *pro/po/cpo*.
2. $\langle A_{proc}, \sqsubseteq_{A_{proc}}, \Sigma_A \rangle$ is a $\Sigma -$ *pro/po/cpo*
3. $C_A = C!_A \cup C?_A$ where $C!_{\overline{A}}$ is a set of monotonic/monotonic/continuous functions $c!._{\overline{A}} : A_{pair} \rightarrow A_{proc}$ and $C?_{\overline{A}}$ is a set of monotonic/monotonic/continuous functions $c?_{\overline{A}} : A_{fun} \longrightarrow A_{proc}$.

We refer to the pair (Σ_A, C_A) as a (Σ, C)-pro/po/cpo structure. An ω-algebraic applicative $(\Sigma, C) -$ *cpo* is called a (Σ, C)-*domain*. $\square$

To reduce the notational complexity we allow us to write pro/po/cpo instead of applicative pro/po/cpo. As we are only dealing with applicative orders this should not cause any confusion.

Definition 2.3 A *pro/po/cpo* homomorphism $\overline{h} : \langle \overline{A}, \sqsubseteq_{\overline{A}} \rangle \longrightarrow \langle \overline{B}, \sqsubseteq_{\overline{B}} \rangle$ is a triplet of mappings, $(h_{proc}, h_{fun}, h_{pair})$, where $h_{proc} : A_{proc} \longrightarrow B_{proc}$, $h_{fun} : A_{fun} \longrightarrow B_{fun}$ and $h_{pair} : A_{pair} \longrightarrow B_{pair}$ are monotonic/monotonic/continuous.

A $(\Sigma, C) -$ *pro/po/cpo homomorphism* $\overline{h} : \langle \overline{A}, \sqsubseteq_{\overline{A}}, \Sigma_{\overline{A}}, C_{\overline{A}} \rangle \longrightarrow \langle \overline{B}, \sqsubseteq_{\overline{B}}, \Sigma_{\overline{B}}, C_{\overline{B}} \rangle$, is a triplet, $(h_{proc}, h_{fun}, h_{pair})$, where $h_{proc} : A_{proc} \longrightarrow B_{proc}$ is a $\Sigma -$ *pro/po/cpo* homomorphism, $h_{fun} : A_{fun} \longrightarrow B_{fun}$ and $h_{pair} : A_{pair} \longrightarrow B_{pair}$ are *pro/po/cpo* homomorphisms and satisfy

$$\begin{array}{l}
h_{proc}(c?_{\overline{A}}.F) = c?_{\overline{B}}.h_{fun}(F) \text{ and } h_{proc}(c!_{\overline{A}}.\Pi) = c!_{\overline{B}}.h_{pair}(\Pi) \\
h_{fun}(F) = h_{proc} \circ F \\
h_{pair}(v, P) = (v, h_{proc}(P))
\end{array}$$

A pro/po/cpo isomorphism and a (Σ, C)-pro/po/cpo isomorphism are defined in the obvious way. □

Notation 2.4 *For* $\overline{A} = (A_1, A_2, A_3)$ *and* $\overline{B} = (B_1, B_2, B_3)$ *we write* $\overline{A} \subseteq \overline{B}$ *if* $A_i \subseteq B_i$ *for* $i = 1, 2, 3$. *If* $\overline{f} = (f_1, f_2, f_3)$ *then we write* $\overline{f} : \overline{A} \longrightarrow \overline{B}$ *for* $f_i : A_i \longrightarrow B_i$, $i = 1, 2, 3$. *We will use a similar pointwise extension of other definitions as well without further explanations.*

We say that

$$\overline{S} = (S_{proc}, S_{fun}, S_{pair}) \subseteq (Proc_{(\Sigma,C)}(PN), Fun_{(\Sigma,C)}(PN), Pairs_{(\Sigma,C)}(PN))$$

is a *recursively closed* subset of $T_{(\Sigma,C)}(PN)$ if whenever $t \in \overline{S}$ then all of its subcomponents are in $\overline{S}$ as well. In this case we write $\overline{S} \subseteq_{rec} T_{(\Sigma,C)}(PN)$. Note that we may apply structural induction when proving properties of recursively closed subsets.

Definition 2.5 Let $\overline{S} \subseteq_{rec} T_{(\Sigma,C)}(PN)$, $\langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}} \rangle$ an applicative $(\Sigma, C)-$*pro* and $PEnv_{\overline{X}}$ the set of process environments $\rho : PN \longrightarrow X_{proc}$. A function

$$\overline{X}[\![\_]\!] = (X_{proc}[\![\_]\!], X_{fun}[\![\_]\!], X_{pair}[\![\_]\!]) : \overline{S} \longrightarrow (PEnv_{\overline{X}} \longrightarrow \overline{X})$$

is an evaluation function if whenever $t \in \overline{S}$ then $\overline{X}[\![t]\!]\rho$ satisfies

$$\begin{array}{rcl}
X_{proc}[\![op(\underline{p})]\!]\rho & = & op_X(X_{proc}[\![\underline{p}]\!]\rho),\ op \in \Sigma \\
X_{proc}[\![c?.f]\!]\rho & = & c?._X X_{fun}[\![f]\!]\rho \\
X_{proc}[\![c!.\pi]\!]\rho & = & c!._X X_{pair}[\![\pi]\!]\rho \\
X_{proc}[\![\ be \rightarrow p_1, p_2]\!]\rho & = & [\![be]\!] \rightarrow (X_{proc}[\![p_1]\!]\rho), (X_{proc}[\![p_2]\!]\rho) \\
X_{fun}[\![[x]p]\!]\rho & = & \lambda v. X_{proc}[\![p[v/x]]\!]\rho \\
X_{pair}[\![(e, p)]\!]\rho & = & ([\![e]\!], X_{proc}[\![p]\!]\rho) \\
X_{proc}[\![X]\!]\rho & = & \rho(X)
\end{array}$$

If A is a *cpo* then, following the standard practice, we may define

$$A_{proc}[\![rec\, X.p]\!]\rho \quad = \quad Y\ \lambda d. A_{proc}[\![p]\!]\rho[d/X]$$

where Y is the least fixed point operator. □

For closed terms the environments do not have any influence on the definition. For terms in $T_{(\Sigma,C)}$ the mapping $\overline{A}[\![ct]\!] = \overline{A}[\![ct]\!]\rho$ may be derived from the definition omitting the last clause of the definition and the occurrence of ρ in the others. We have the following result.

Theorem 2.6 *Let $\overline{S} = (S_{proc}, S_{fun}, S_{pair}) \subseteq_{rec} T_{(\Sigma,C)}(PN)$ and $\langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}}\rangle$ be an applicative $(\Sigma, C) - pro$. Then there is at most one evaluation mapping*

$$\overline{X}[\![\_]\!] = (X_{proc}[\![\_]\!], X_{fun}[\![\_]\!], X_{pair}[\![\_]\!]) : \overline{S} \longrightarrow (PEnv_{\overline{X}} \longrightarrow \overline{X})$$

If $\langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}}\rangle$ is fully applicative then such an evaluation mapping exists.

2.3 Properties Derived From the Compact Elements

In this section we will describe how we can turn an algebraic *cpo* into a (Σ, C)-domain if we have a complete description of its compact elements. So let us assume that $\langle \overline{B}, \sqsubseteq_{\overline{B}}\rangle$ is an algebraic *cpo*. Further assume that $\langle \overline{X}, \prec_{\overline{X}}\rangle$ is a preorder where the induced partial order $\langle \overline{X}/\sim, \prec_{\overline{X}/\sim}\rangle$, $\sim = \prec \cap \prec^{-1}$ is isomorphic to $\langle Co(\overline{B}), \sqsubseteq_{Co(\overline{B})}\rangle$ under $\overline{\phi} : \langle \overline{X}/\sim, \prec_{\overline{X}}\rangle \longrightarrow \langle Co(\overline{B}), \sqsubseteq_{Co(\overline{B})}\rangle$. By defining a monotonic (Σ, C) structure on $\langle \overline{X}, \prec_{\overline{X}}\rangle$ we get automatically an induced monotonic structure on $\langle \overline{X}/\sim, \prec_{\overline{X}/\sim}\rangle$ and thereby on the isomorphic $\langle Co(\overline{B}), \sqsubseteq_{Co(\overline{B})}\rangle$. Now, because of the algebraicity this structure may be extended in a unique way to a continuous structure on $\langle \overline{B}, \sqsubseteq_{\overline{B}}\rangle$. This procedure may be described formally by the following Theorem. We note that the function $\overline{\psi} = \overline{inc} \circ \overline{\phi} \circ [\_]$ where $\overline{inc} : \langle Co(\overline{B}), \sqsubseteq_{Co(\overline{B})}\rangle \longrightarrow \langle \overline{B}, \sqsubseteq_{\overline{B}}\rangle$ is the inclusion mapping and $[\_] : \langle \overline{X}, \prec_{\overline{X}}\rangle \longrightarrow \langle \overline{X}/\sim, \preceq_{\overline{X}/\sim}\rangle$ is the quotient mapping, is a *po*-homomorphism.

Theorem 2.7 *Assume that $\langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}}\rangle$ is a $(\Sigma, C) - pro$ with $\sim = \prec_{\overline{X}} \cap \prec_{\overline{X}}^{-1}$ and the induced partial order $\langle \overline{X}/\sim, \preceq_{\overline{X}/\sim}\rangle$. Further assume that $\langle \overline{B}, \sqsubseteq_{\overline{B}}\rangle$ is an applicative algebraic cpo where the po of compact elements is isomorphic to $\langle \overline{X}/\sim, \preceq_{\overline{X}/\sim}\rangle$ under the isomorphism*

$$\phi : \langle \overline{X}/\sim, \preceq_{\overline{X}/\sim}\rangle \longrightarrow \langle Co(\overline{B}), \sqsubseteq_{Co(\overline{B})}\rangle$$

Then the following holds:

1. *There exists a unique (Σ, C)-cpo structure, $(\Sigma_{\overline{B}}, C_{\overline{B}})$ which extends $\langle \overline{B}, \sqsubseteq_{\overline{B}}\rangle$ to a (Σ, C)-domain and extends $\overline{\psi} = \overline{inc} \circ \overline{\phi} \circ [\_]$ to a (Σ, C)-homomorphism. The structure $(\Sigma_{\overline{B}}, C_{\overline{B}})$ preserves compactness.*

2. *Let $\overline{B}[\![\_]\!] : T^{rec}_{(C,\Sigma)}(X) \longrightarrow \overline{B}$ be an evaluation mapping. If $S \subseteq_{rec} T_{(C,\Sigma)}$ and $X[\![\_]\!] : S \longrightarrow X$ is an evaluation mappings then for all $t \in \overline{S}$:*

$$\overline{B}[\![t]\!] = \overline{\psi}(X[\![t]\!]).$$

3 The Language Late-*CCS* and its Semantics

In this section we will give a concrete language and define a denotational model for it. We also give an equationally based proof system and show its soundness and completeness with respect to the denotational model.

3.1 A Domain Equation for Applicative Communication Trees

The language *Late-CCS*, is a modification of the original CCS in the spirit of the late approach. As described in the Introduction it is basically a sublanguage of the more general π-calculus in which only simple values are transmitted in communications whereas the latter allows port names to be transmitted as well as simple values. Described in our general framework Late-CCS is obtained by instantiating the signature Σ by the standard operator of CCS. So we let Σ consist of the nullary operators NIL and Ω, the families of unary operators $\_ \setminus c, c \in C$ and $\_[R]$ where R is a finite permutation of the process names, and the binary operators $+$ and $|$. For the motivation of these operators we refer to the standard theory of CCS in [Mil89].

Next we construct a (Σ, C)-domain to give a denotational semantics for our language. The definition of this model is motivated by the following existing work.

In 1979 Milne and Milner [MM79], gave a domain theoretical definition of the concept of *communicating processes*. This definition reflects the late semantic approach described above. Each process has a collection of typed ports through which it may communicate with other processes. There are two types of communications; input and output. If we abstract from the types then input capability of a process p along a channel c is modelled as an element of the domain $V \longrightarrow P$ labelled by the channel name c, where the domain of processes is denoted by the *cpo* P and the domain of values by V. An output capability of p on c, on the other hand, is modelled as an element of $V \times P$ labelled by c. A process is modelled as a set of communication capabilities or more precisely as an element of the Smyth power-domain [Sm78], over the domain of communication capabilities. The empty set is embedded into the domain in such a way that it becomes the top element of the domain. This leads to a recursive definition of the process domain which is defined as the initial solution of corresponding domain equation.

In [Abr92] Abramsky pointed out a disadvantage of this model: the use of the Smyth power-domain to model communicating processes rules out the possibility of any correspondence with bisimulation. Also the embedding of the empty set (which corresponds to the inactive and convergent process) as the top element of the model is intuitively incorrect. In the same reference the author defines a model to describe the semantics of pure processes. This model is similar to the model of [MM79] and is also obtained as the initial solution to a recursive domain equation. The main difference is that Abramsky defines his model in terms of the Plotkin power-domain instead of the Smyth power-domain. He adds the empty set to the model as an isolated element only comparable with itself and the bottom element of the model in the obvious way. He then interprets the calculus $SCCS$ in the model and shows the full abstractness of this interpretation with respect to strong ω-prebisimulation.

The model we define is basically the model of [MM79] where the modifications of Abramsky's are adapted. Thus we define a model which models value-passing based on the late approach using the Plotkin power-domain with the empty set adjoined as an isolated element.

Here the main difference is that we use a different representation for Plotkin power-domain to the one used in [Abr92]. The representation we use is the one due to Smyth, [Sm78] and may be described as follows:

Let $\langle A, \sqsubseteq\rangle$ be an ω-algebraic *cpo* and $M[A]$ the family of finite, non-empty sets of compact elements of A. The Egli-Milner preorder on $M[A]$, $\sqsubseteq_{EM(A)}$, is defined by

$$X \sqsubseteq_{EM(A)} Y \text{ if } \forall x \in X \exists y \in Y. x \sqsubseteq_A y \text{ and } \forall y \in Y \exists x \in Y. y \sqsubseteq_A x$$

The Plotkin power-domain of $\langle A, \sqsubseteq\rangle$, $\langle P[A], \sqsubseteq_{P[A]}\rangle$ is the ideal completion of the preorder $\langle M[A], \sqsubseteq_{EM(A)}\rangle$. From the standard theory, [Plo81b], we know that $\langle P[A], \sqsubseteq_{P[A]}\rangle$ is an ω-algebraic *cpo* and the *po* of compact elements is isomorphic to the *po* derived from the preorder $\langle M[A], \sqsubseteq_{M[A]}\rangle$. Given an ω-algebraic *cpo* we write $P^0[D]$ for the Plotkin power-domain over D with the empty set adjoined as an isolated element in the preorder. More precisely the elements of $P^0[D]$ are given by $P[D] \cup \{\emptyset\}$ with the order:

$$X \sqsubseteq_{P^0[D]} Y \quad \text{if} \quad \begin{array}{rl} & X, Y \in P[D] \text{ and } X \sqsubseteq_{P[D]} Y \\ \text{or} & Y = \{\emptyset\} \text{ and } (X = \{\emptyset\} \text{ or } X = \bot) \end{array} \tag{1}$$

Now we can give the following definition:

Definition 3.1 [Applicative Communication Trees] Let $Chan$ (the set of channels) and Val (the set of values) be countable sets and $Act = \{c?|c \in Chan\} \cup \{c!|c \in Chan\} \cup \{\tau\}$ (the set of actions). We define the applicative *cpo* of applicative communication trees, $\langle \overline{ACT}, \sqsubseteq\rangle$, as follows: $\langle ACT, \sqsubseteq\rangle$ is the initial solution in the category of cpos with embeddings of the recursive domain equation:

$$D = P^0[\sum_{e \in Act} D_e]$$

where $D_{c?} = Val \to D$, $D_{c!} = Val \times D$ and $D_\tau = D$. Then we define $ACT_{proc} = ACT$, $ACT_{fun} = Val \longrightarrow ACT$ and $ACT_{pair} = Val \times ACT$ with the usual induced order. $\langle \overline{ACT}, \sqsubseteq_{\overline{ACT}}\rangle$ is a fully applicative *cpo* which we refer to as ACT. □

By unfolding the recursive definition of ACT, we get a description of the compact elements from the standard theory. Thus we define

$$Co(ACT) = \bigcup_{n=0}^{\infty} Comp^n$$

where

$$Comp^0 = \{\bot\}$$

and

$$\begin{aligned} Comp^{n+1} &= Co(P^0[\textstyle\sum_{e \in Act}(Comp^n)_e]) \\ &= M[\textstyle\sum_{e \in Act}(Comp^n)_e] \cup \{\emptyset\} \end{aligned}$$

where $(Comp^n)_{c?} = Val \longrightarrow Comp^n$, $(Comp^n)_{c!} = Val \times Comp^n$ and $(Comp^n)_\tau = Comp^n$. We recall that for an algebraic *cpo* A, $M[A]$ is defined as the family of non-empty sets of compact elements of A. The empty set is added to the family $Comp^n$ as we are using the power-domain operator P^0 rather than P. Defining $Co(ACT)$ this way and ordering it by $\sqsubseteq^0_{EM}$, the Egli-Milner preorder over $Co(ACT)$ extended as described above, gives a representation of the compact elements of the model ACT. This suggests an inductive definition to describe the compact elements and the preorder on these. Thus we define the set K and the preorder $\prec$ on it inductively and prove that $\langle K, \prec\rangle$ is a representation of $\langle Co(ACT), \sqsubseteq^0_{EM}\rangle$

Definition 3.2 We define K as the least set which satisfies

1. $\emptyset \in K$
2. $\{\perp\} \in K$
3. $c \in Chan, V \subseteq_{fin} Val$ and $\forall v \in V. k_v \in K$ implies $\{\langle c?, \lambda v.x \in V \longrightarrow k_v, \Omega\rangle\} \in K$
4. $c \in Chan, v \in Val, k \in K$ implies $\{\langle c!, (v, k)\rangle\} \in K$
5. $k \in K$ implies $\{\langle \tau, k\rangle\} \in K$
6. $k_1, k_2 \in K$ implies $k_1 \cup k_2 \in K$

The preorder $\prec$ is defined as the least preorder on K which satisfies

1. $\{\perp\} \prec \emptyset$
2. $k_1 \prec k_2$ if $\forall a \in k_1 \exists b \in k_2. a \leq b$ and $\forall b \in k_1 \exists a \in k_2. a \leq b$ where $\leq$ is defined on the elements of the sets in K by
   (a) $\forall a. \perp \leq a$
   (b) $\langle \tau, k\rangle \leq \langle \tau, k'\rangle$ if $k \prec k'$
   (c) $\langle c?, f\rangle \leq \langle c?, g\rangle$ if $\forall v \in Val. f(v) \prec g(v)$
   (d) $\langle c!, (v, k)\rangle \leq \langle c!, (v, k')\rangle$ if $k \prec k'$

We let $\approx = \prec \cap \prec^{-1}$. □

Proposition 3.3 $\langle K/\approx, \prec_{K/\approx}\rangle = \langle Co(ACT), \sqsubseteq^0_{EM}\rangle$ *(where we also use* $\sqsubseteq^0_{EM}$ *for the partial order induced by* $\sqsubseteq^0_{EM}$ *on* $Co(ACT)$*).*

Definition 3.4 We define $\langle \overline{K}, \prec_{\overline{K}}\rangle$ by $K_{proc} = K$, $K_{fun} = Fin(Val \longrightarrow K)$ and $K_{pair} = Val \times K$ and $\prec_{\overline{K}}$ is the induced preorder. □

3.2 Definition of the Operators in the Model

In this subsection we will define the operators in $\Sigma_{\overline{ACT}}$ and prove their continuity. In the definitions we take advantage of Theorem 2.7. Thus we only have to define the operators on the applicative *pro* $\langle \overline{K}, \prec_{\overline{K}}\rangle$ and make sure that they are monotonic.

Definition 3.5 We define $\Sigma_{\overline{K}}$ as follows:

$$\begin{array}{lrcl}
\textbf{Constants:} & NIL_{\overline{K}} & = & \emptyset \\
& \Omega_{\overline{K}} & = & \{\bot\} \\
\textbf{Prefixing:} & c?_{\overline{K}}._ & = & \lambda f.\{\langle c?, f\rangle\} \\
& c!_{\overline{K}}._ & = & \lambda(v,k).\{\langle c!, (v,k)\rangle\} \\
& \tau_{\overline{K}}._ & = & \lambda k.\{\langle \tau, k\rangle\}
\end{array}$$

Nondeterminism: $+_{\overline{K}} \;=\; \cup$

Restriction: $\_\backslash c_{\overline{K}} = F_c$ where $F_c : K_{proc} \to K_{proc}$ is defined by

$$\begin{array}{rcl}
F_c\{\bot\} & = & \{\bot\} \\
F_c(\emptyset) & = & \emptyset \\
F_c\{\langle b?, f\rangle\} & = & \begin{cases} \{\langle b?, F_c \circ f\rangle\} & \text{if } b \neq c \\ \emptyset & \text{otherwise} \end{cases} \\
F_c\{\langle b!, (v,k)\rangle\} & = & \begin{cases} \{\langle b!, (v, F_c k)\rangle\} & \text{if } b \neq c \\ \emptyset & \text{otherwise} \end{cases} \\
F_c\{\langle \tau, k\rangle\} & = & \{\langle \tau, F_c k\rangle\} \\
F_c(k_1 \cup k_2) & = & (F_c k_1) \cup (F_c k_2)
\end{array}$$

Renaming: $\_[R]_{\overline{K}} = G_R$ where $G_R : K_{proc} \to K_{proc}$ is defined by

$$\begin{array}{rcl}
G_R\{\bot\} & = & \{\bot\} \\
G_R\emptyset & = & \emptyset \\
G_R\{\langle c?, f\rangle\} & = & \{\langle R(c)?, G_R \circ f\rangle\} \\
G_R\{\langle c!, (v,k)\rangle\} & = & \{\langle R(c)!, (v, G_R k)\rangle\} \\
G_R\{\langle \tau, k\rangle\} & = & \{\langle \tau, G_R k\rangle\} \\
G_R(k_1 \cup k_2) & = & (G_R(k_1)) \cup (G_R(k_2))
\end{array}$$

Parallel Composition: $\_ \,|_{\overline{K}}\, \_ = F$ where

$$\begin{array}{rcl}
F & = & int \cup comm \cup div \\
int & = & int_{in} \cup int_{out} \cup int_{\tau} \\
int_{in}(x,y) & = & \{\langle c_x?, \lambda v.F(f_x(v)), y\rangle | \langle c_x?, f_x\rangle \in x\} \\
& \cup & \{\langle c_y?, \lambda v.F(x, f_y(v))\rangle | \langle c_y?, f_y\rangle \in y\} \\
int_{out}(x,y) & = & \{\langle c_x!, (v, F(x', y)\rangle | \langle c_x!, (v, x')\rangle \in x\} \\
& \cup & \{\langle c_y!, (v, F(x, y')\rangle | \langle c_y!, (v, y')\rangle \in y\} \\
int_{\tau} & = & \{\langle \tau, F(x', y)\rangle | \langle \tau, x'\rangle \in x\} \\
& \cup & \{\langle \tau, F(x, y')\rangle | \langle \tau, y'\rangle \in y\} \\
comm(x,y) & = & \{\langle \tau, F(f(v), y')\rangle \mid \exists c, v. \langle c?, f\rangle \in x \text{ and } \langle c!, (v, y')\rangle \in y\} \\
& = & \{\langle \tau, F(x', g(v))\rangle \mid \exists c, v. \langle c?, g\rangle \in y \text{ and } \langle c!, (v, x')\rangle \in x\}
\end{array}$$

$$div(x,y) \;=\; \begin{cases} \{\perp\} & \text{if } \perp \in x \cup y \\ \emptyset & \text{otherwise} \end{cases}$$

□

Lemma 3.6 *$\langle \overline{K}, \prec_{\overline{K}}, \Sigma_{\overline{K}}, C_{\overline{K}} \rangle$ is a (Σ, C)-pro.*

We finish this section by summarizing our results. This is done by the following theorem.

Theorem 3.7 *$\langle \overline{ACT}, \sqsubseteq_{\overline{ACT}}, \Sigma_{\overline{ACT}}, C_{\overline{ACT}} \rangle$, where $(\Sigma_{\overline{ACT}}, C_{\overline{ACT}})$ is the (unique) (Σ, C)-po structure induced by $\langle \overline{K}, \prec_{\overline{K}}, \Sigma_{\overline{K}}, C_{\overline{K}} \rangle$, is a fully applicative (Σ, C)-domain.*

3.3 Syntactically Compact Elements

In this last subsection we will show that the compact elements of the model $\overline{ACT}$ may be denoted in our syntax. For this purpose we introduce the *syntactically compact Terms*, $CoTerms = (CoProc, CoFun, CoPair)$.

As usual syntactically finite terms are those without occurrences of recursion. We define compact terms as the syntactically finite ones which only use a finite number of values in a nontrivial way. Note that in this case the number of channels used by the term is automatically finite. We start by introducing some notation.

Notation 3.8 *Let $\underline{v_n} = (v_1, \ldots, v_n)$ and $\underline{p}_n = (p_1, \ldots, p_n)$. We write $x : \underline{v_n} \longrightarrow \underline{x_n}$ for $x = v_1 \longrightarrow p_1, (x = v_2 \longrightarrow p_2, (\ldots x = v_n \longrightarrow p_n, \Omega) \ldots)$. Further we let $\{\underline{v_n}\} = \{v_i | \underline{v_n} = (v_1, \ldots, v_n)\}$ and similarly for $\{p_n\}$.*

Definition 3.9 [Compact Terms] The set of compact terms, $CoTerms$, is the triplet $(CoProc, CoFun, CoPairs)$ where the sets $CoProc$, $CoFun$ and $CoPairs$ are the least sets satisfying:

1. $NIL, \Omega \in CoProc$
2. $\overline{p} \in CoProc$ implies $op(\overline{p}) \in CoProc$, $op = |, +, \_\backslash, \_[R], \tau.\_$
3. $\pi \in CoPair$, $c \in Chan$ implies $c!.\pi \in CoProc$
4. $f \in CoFun$ and $c \in Chan$ implies $c?.f \in CoProc$
5. $p \in CoProc$ and $e \in Exp$ implies $(e, p) \in CoPairs$
6. $\{\underline{p_n}\} \subseteq CoProc$, $\{\underline{v_n}\} \subseteq Val$ and $x \in Var$ implies $[x].\, x : \underline{v_n} \longrightarrow \underline{p_n} \in CoFun$.

We say that a term is *compact* if it belongs to $CoTerms$. We let Cot, Cop, etc. range over $CoTerms$, $CoProc$, etc.

□

Note that $CoTerms = (CoProc, CoFun, CoPair) \subseteq_{rec} (Proc, Fun, Pairs)$. We have the following:

Theorem 3.10

1. *$\overline{ACT}[\![\_]\!]$ and $K[\![\_]\!]$ are well-defined and for all $t \in CoTerms$: $\overline{ACT}[\![t]\!]\rho = [\overline{K}[\![t]\!]]_{\approx}$ where $[\_]_{\approx}$ is the quotient mapping with respect to $\prec$.*

2. *For any $Cot \in CoTerms$ $\overline{ACT}[\![Cot]\!] \in Co(\overline{ACT})$*

3. *For all $k \in Co(\overline{ACT})$ there is a $Cot \in CoTerms$ such that $\overline{ACT}[\![Cot]\!] = k$.*

4 Algebraic Laws and Proof Systems

In this section we will introduce a proof system supported by the model $\overline{ACT}$. We proceed by introducing first a system, E, for finite processes which we then extend to a system, E_{rec}, which takes care of recursive processes. We prove the soundness and completeness of E_{rec} with respect to the model.

The proof system E is equationally based where the equations reflect naturally the properties of the operators in the model.

The inference rules describe the structure and the preorder in the model and their interaction with the operators. Because of the two level structure of our syntax, we have the equations

$$\begin{array}{ll} (res\, fun) & ([x]X) \setminus c = [x](X \setminus c) \\ (res\, pair) & (e, X) \setminus c = (e, X \setminus c) \\ (ren\, fun) & ([x]X)[R] = [x](X[R]) \\ (ren\, pair) & (e, X)[R] = (e, X[R]) \end{array}$$

and the rules

$$\begin{array}{ll} (fun) & \dfrac{p[v/x] \sqsubseteq q[v/x] \text{ for every } v \in V}{[x]p \sqsubseteq [x]q} \\ (pair) & \dfrac{[\![e_1]\!] = [\![e_2]\!], p \sqsubseteq q}{(e_1, p) \sqsubseteq (e_2, q)} \end{array}$$

The extended system E_{rec} is then obtained by adding to E three new rules, $(rec1)$, $(rec2)$ and $(\omega - rule)$, to take care of recursion. These rules are all fairly standard. The compact approximations, $p^{(n)}$, used in the rule $(\omega - rule)$ are defined in Definition 4.1.

In the rule $(res\ pre)$ we use the notation $chan(pre)$, where $chan(c?) = chan(c!) = c$. In the interleaving law the summation notation is justified by equations (+1)-(+4) and an empty sum is understood as NIL. $\{+\Omega\}$ indicates that Ω is an optional summand of a term and Ω is a summand of the right hand side if it is either a summand of X or Y. To simplify the notation we assume that i, j etc. in the sums $\sum_i$, $\sum_j$, etc. range over finite index sets I, J, etc.

We now define the compact approximations used in the ω-rule of the proof system.

$$
\begin{array}{lrcl}
(+1) & X+(Y+Z) & = & (X+Y)+Z \\
(+2) & X+Y & = & Y+X \\
(+3) & X+X & = & X \\
(+4) & X+NIL & = & X \\
(res+) & (X+Y)\setminus c & = & X\setminus c+Y\setminus c \\
(res\,pre) & (a.X)\setminus c & = & \begin{cases} a.(X\setminus c) & \text{if } c\neq chan(pre) \\ NIL & \text{otherwise} \end{cases} \\
(res\,fun) & ([x]X)\setminus c & = & [x](X\setminus c) \\
(res\,pair) & (e,X)\setminus c & = & (e,X\setminus c) \\
(res\,NIL) & NIL\setminus c & = & NIL \\
(res\,div) & \Omega\setminus c & = & \Omega \\
(ren+) & (X+Y)[R] & = & X[R]+Y[R] \\
(ren\,pre) & (a.X)[R] & = & R(a).X[R] \\
(ren\,fun) & ([x]X)[R] & = & [x](X[R]) \\
(ren\,pair & (e,X)[R] & = & (e,X[R]) \\
(ren\,NIL) & NIL[R] & = & NIL \\
(ren\,div) & \Omega[R] & = & \Omega \\
(NIL\,par) & NIL\mid X & = & X\mid NIL \;=\; X \\
(div) & \Omega & \sqsubseteq & X
\end{array}
$$

Let $X = \sum_i \tau.X_i + \sum_j a_j?.[x]X'_j + \sum_k a'_k!.(v_k, X''_k)\{+\Omega\}$ and $Y = \sum_l \tau.Y_l + \sum_m b_m?.[y]Y'_m + \sum_n b'_n!.(v_n, Y''_n)\{+\Omega\}$. Then

$$X \mid Y = INTL(X,Y) + COMM(X,Y)\{+\Omega\}$$

where

$$INTL(X,Y) = INTL_\tau(X,Y) + INTL_{in}(X,Y) + INTL_{out}(X,Y)$$

where

$$
\begin{array}{l}
INTL_\tau(X,Y) = \sum_i \tau.(X_i|Y) + \sum_l \tau.(X|Y_l) \\
INTL_{in}(X,Y) = \sum_j a_j?.[x](X'_j|Y) + \sum_m b_m?.[y](X|Y'_m) \\
INTL_{out}(X,Y) = \sum_k a'_k!.(v_k, X''_k|Y) + \sum_n b'_n!.(v'_n, X|Y''_n)
\end{array}
$$

and

$$COMM(X,Y) = \sum_{j,n:a_j=b'_n} \tau.X'_j[v_n/x]|Y''_n + \sum_{k,m:a'_k=b_m} \tau.X''_k|Y'_m[v_k/y]$$

Figure 1: Equations

$$(ref) \qquad p \sqsubseteq p \qquad\qquad (trans) \qquad \frac{p \sqsubseteq q,\ q \sqsubseteq r}{p \sqsubseteq r}$$

$$(sub) \qquad \frac{p_i \sqsubseteq q_i}{op(\underline{p}) \sqsubseteq op(\underline{q})} \qquad op \in \sum$$

$$(pre) \qquad \frac{p \sqsubseteq q}{\mu.p \sqsubseteq \mu.q}$$

$$(rec1) \qquad \frac{p \sqsubseteq q}{recP.p \sqsubseteq recP.q} \qquad\qquad (rec2) \qquad \frac{}{recP.p = p[recP.p/P]}$$

$$(\omega - rule) \qquad \frac{p^{(n)} \sqsubseteq q \text{ for all } n}{p \sqsubseteq q}$$

$$(inst) \qquad \frac{}{p\sigma \sqsubseteq q\sigma} \text{ for every inequation } p \sqsubseteq q \qquad \text{and instantiation } \sigma$$

$$(cond1) \qquad \frac{[\![be]\!] = T}{be \longrightarrow p, q = \ p} \qquad\qquad (cond2) \qquad \frac{[\![be]\!] = F}{be \longrightarrow p, q = \ q}$$

$$(pair) \qquad \frac{[\![e]\!] = [\![e']\!], p \sqsubseteq q}{(e, p) \sqsubseteq (e', q)}$$

$$(fun) \qquad \frac{p[v/x] \sqsubseteq q[v/x] \text{ for every } v \in V}{[x]p \sqsubseteq [x]q}$$

$$(\alpha - red) \qquad \frac{}{[x]p = [y]p[y/x]} \qquad \text{if } y \text{ not free in } p$$

Figure 2: The Proof System E_{rec}

Definition 4.1 [Compact Approximations] The n-th compact approximation of a term is defined inductively by following:

I. $i) p^0 = \Omega$

$ii)$ $1. P^{(n+1)} = P$

$2. (op(\underline{p}))^{(n+1)} = op(\underline{p}^{(n+1)})$

$3. (\mu.u)^{(n+1)} = \mu.u^{(n+1)}$

$5. (recP.p)^{(n+1)} = p^{(n+1)}[(recP.p)^{(n)}/P]$

$6. (be \longrightarrow p, u)^{(n+1)} = be \longrightarrow p^{(n+1)}, u^{(n+1)}$

II. $([x]p)^{(n+1)} = [x](x \in V_{n+1} \longrightarrow p^{(n+1)}, \Omega)$

III. $((v, p))^{(n+1)} = (v, p^{(n+1)})$

□

The compact approximations have the following property:

Theorem 4.2 *For all n and all $ct \in CTerms$*

1. $ct^{(n)} \in CoTerms$
2. $[\![ct]\!]\rho = \bigsqcup_n [\![ct^{(n)}]\!]\rho$

We will end this section by stating the soundness and completeness of the proof system E_{rec} with respect to the denotational semantics. For the proof we refer to the full version of the paper.

Theorem 4.3 [Soundness and Completeness] *For all closed terms ct, cu in $CTerms$, $ct \sqsubseteq_{E_{rec}} cu$ if and only if $\overline{\mathrm{ACT}}[\![ct]\!] \sqsubseteq \overline{\mathrm{ACT}}[\![cu]\!]$, i.e. the proof system E_{rec} is sound and complete with respect to the denotational semantics.*

5 Conclusion

In the first part of this paper we have set up a general framework for describing the syntax and the denotational semantics for value-passing calculi which support the late approach. For this purpose the standard notion of signature and Σ-algebras and -orders have been extended to the so-called applicative signatures and (Σ, C)-algebras and -orders. Furthermore we show how we may take an advantage of the ω-algebraicity of a model to define the operators in the model.

In the second part of the paper we define the language Late-CCS which is a modification of the standard CCS with values due to the late semantic approach. This language is basically the π-calculus where the values allowed are restricted to be of the simple type only.

A denotational model for Late-CCS is defined, an instantiation of the general class of models we defined. The carrier set of this model is an ω-algebraic cpo and is obtained as a solution to an recursive domain equation and is a direct extension of the model defined by Abramsky in [Abr92].

The operators are constructed by isolating the compact elements, define the operators in the derived partial order, make sure that they are monotonic which then allows us to extend them to continuous functions defined the whole *cpo*.

In the last subsection we define a equationally based proof system and show its soundness and completeness with respect to the model.

As already explained the construction of the domain equation and thereby the definition of the model is strongly inspired by the idea of bisimulation preorder. In the accompanying paper [Ing94a] it is shown that the original late bisimulation is too strong to meet the preorder of the model ACT. The algebraicity of the model implies that the preorder in the model is completely decided by the compact elements. Behaviourally this can be interpreted as that the preorder may be obtained by finite observations. This is not the case for bisimulation as well known even for the pure calculus. In [Ing94a] a finitary version of the preorder is defined by mimicking the ω-algebraicity of the model on the syntactical level. This preorder is shown to coincide with the preorder in the model in the sense that the model is fully abstract with respect to this preorder.

References

[Abr92] S. Abramsky: A Domain Equation For Bisimulation, *Information and Computation*, Vol. 92, pp 161-218, 1992.

[EN86] Engberg, U., Nielsen, M., “A Calculus of Communicating Systems with Label-passing”, Report DAIMI PB-208, Computer Science Department, University of Aarhus, 1986.

[Gue81] Guessarian, I., “Algebraic Semantics”, *Springer-Verlag Lecture Notes in Computer Science*, vol.99, 1981.

[HI93a] Hennessy, M., Ingólfsdóttir, A. A Theory of Communicating Processes With Value - Passing, *Information and Computation*, Vol. 107, no. , pp 202-236, 1993.

[HM85] M. Hennessy , R. Milner: *Algebraic Laws for Nondeterminism and Concurrency*, Journal of the Association for Computing Machinery, vol. 32, No. 1, pp. 137-161, 1985.

[HP80] Hennessy, M. and Plotkin, G., “A Term Model for CCS”, *Springer-Verlag Lecture Notes in Computer Science*, vol.88, 1980.

[He88] Hennessy, M. *Algebraic Theory of Processes.* MIT Press, Cambridge, 1988.

[Hoa78] Hoare, C.A.R. “Communicating Sequential Processes”, *Comm. ACM*, 21(8), 666-677 1978.

[Ing93] Ingólfsdóttir, A. Late and Early Semantics Coincide for Testing, Aalborg University Technical Report, IR 93-2008, 1993. To appear in *Theoretical Computer Science.*

[Ing94] Ingólfsdóttir, A., Semantic Models for Communicating Processes with Value-passing, Ph.D Thesis, Technical Report 8/94, Sussex University, 1994.

[Ing94a] Ingólfsdóttir, A., Semantic Theory for Value-Passing Processes, Late Approach, Part II: Behavioural Semantics and Full Abstractness. In preparation.

[IT92] A. Ingólfsdóttir, B. Thomsen. Semantic Models for *CCS* with Values. Technical Report 63, Programming Methodology Group, Chalmers University of Technology, 1992. In Proceedings of the Workshop on Concurrency.

[MM79] G. Milne , R. Milner: *Concurrent Processes and Their Syntax*, Journal of the Association for Computing Machinery, Vol. 26, No. 2, pp. 302-321, 1979.

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

[Mil83] Milner, R. *Calculi for Synchrony and Asynchrony*, Theoretical Computer Science, 25, pp 267–310, 1983

[Mil89] Milner, R., *Communication and Concurrency*, Prentice-Hall, London 1989.

[Mil91] R. Milner, Polyadic π-Calculus. A tutorial. To appear in *Proceedings of the International Summer School on Logic and Algebra of Specifications*, Marktoberdorf, 1991.

[MPW89] R. Milner, J. Parrow & D. Walker, "A Calculus of Mobile Processes" Part I+II, *Information and Computation*, Vol. 100 no. 1, pp 1-77, 1992.

[MPW91] R. Milner, J. Parrow & D. Walker, *Modal Logics for Mobile Processes* Proceedings of Concur '91 , Lecture Notes in Computer Science 327, pp. 45-60, Springer-Verlag, 1991.

[Plo76] G. Plotkin: *A Powerdomain Construction*, SIAM Journal on Computing, 5 (1976), pp. 452-487, 1976.

[Plo81b] Plotkin, G., "Lecture Notes in Domain Theory", University of Edinburgh, 1981.

[Sm78] Smyth, M., Powerdomains, *JCSS* 16(1), 1978

[SP 82] Smyth, M. and Plotkin, G., "The Category-Theoretic Solution of Recursive Domain Equations", *SIAM Journal on Computing*, vol.11, No.4, 1982.

[Win85] Winskel, G., On Powerdomains and Modality, *Theoretical Computer Science*, 36, pp. 127- 137, 1985.