Basic Research in Computer Science

BRICS RS-95-3 A. Ingólfsdóttir: Value–Passing Processes, Late Approach, Part I

A Semantic Theory for Value–Passing Processes Late Approach

Part I: A Denotational Model and Its Complete Axiomatization

Anna Ingólfsdóttir

BRICS Report Series **RS-95-3**

ISSN 0909-0878 **January 1995**

**Copyright © 1995, BRICS, Department of Computer Science
University of Aarhus. All rights reserved.**

**Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark**

**Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@daimi.aau.dk**

A Semantic Theory for Value–Passing Processes Late Approach

Part I: A Denotational Model and Its Complete Axiomatization

Anna Ingólfsdóttir

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

Abstract

A general class of languages and denotational models for value-passing calculi based on the late semantic approach is defined. A concrete instantiation of the general syntax is given. This is a modification of the standard *CCS* according to the late approach. A denotational model for the concrete language is given, an instantiation of the general class. An equationally based proof system is defined and shown to be sound and complete with respect to the model.

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 process which is ready to input a value on a channel c (e.g. a prefixing with an input action, $\overline{c}(x).p$) is interpreted as a non-deterministic choice between pure terms of the form $\overline{c}_v.p[v/x]$, 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 more recent work on the π-calculus, [MPW92], 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 output 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 input

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

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 output counterpart in the communication provides. Symmetrically the result of reporting the willingness to output an uninterpreted value on the channel c is given by the action c. By performing this action the process evolves to a term which basically consists of a data expression, i.e. the expression whose value the sender wants to output, and a process expression, i.e. what remains to be executed of the sender.

In a more recent version of the π-calculus, the Polyadic π-calculus presented in [Mil91], the outcomes of input and output actions are modelled by extending the syntax with the new constructions *abstractions* and *concretions*.

The semantics for Thomsen's *plain CHOCS* in [Tho89] is based on the late approach although the author does not give it a specific name.

In the literature the late semantic approach has been investigated in different ways, both in connection with the π-calculus and higher order calculi (see e.g. [MPW91, Hen94, San93]) and also with the main focus on the simpler case where only first order values are allowed (see e.g. [HL93a, HL93b, Ing94, Ing93]).

In this series of two companion papers we will try to contribute to the studies of the late semantics of communicating processes. We will concentrate on processes which allow transmission of simple values only. Of course studying value-passing processes is interesting in itself, but we also believe that it may give some insight into the nature of the late approach which may be useful in future studies of the semantics of the more complicated calculi of higher order or mobile processes (such as the π-calculus).

To make our studies more complete we follow the line of [Hen88a] and [HI93] and introduce a trinity of semantic descriptions for a CCS like process language and show their equivalence. More precisely we give an operational or behavioural semantics in terms of an extended version of labelled transition systems and corresponding bisimulation based relations, axiomatic semantics by means of an equationally based proof system and denotational semantics following the Scott-Strachey approach. Like many researchers in the area of process algebra we believe that the operational or the behavioural semantic model is the most natural and intuitive one, but that different kinds of semantic descriptions give important alternative views of the nature of the interpretation of process languages. For instance the interpretation of an infinite process modelled by an algebraic cpo is fully specified by the interpretation of its finitely computable approximations. This is not the case for many behaviourally based semantics as will be explained in more detail in the sequel to this paper, [Ing95a].

One of the main purposes of this series of papers is to give an operational characterization of the denotational interpretation of a value-passing process in an algebraic cpo. Therefore we start by giving a denotational characterization of value-passing processes using the late principle. We also give an equationally based proof system which can be naturally derived from the denotational semantics and show its soundness and completeness with respect to the denotational semantics. This is the content of this paper. Its sequel, [Ing95a], is devoted to defining operational semantics, analysing operationally

the denotational semantics and to define a reasonable behavioural relation between processes which characterizes the relation induced by the denotational semantics. All three models are based on the idea of bisimulation.

In this paper, Part I in the series, we will develop a semantic theory for processes with values based on the idea of (strong) bisimulation with emphasis on the late approach. The semantics will be denotational and we shall follow the Scott-Strachey approach. Our development will proceed in two steps. First we describe a general theory for denotational semantics of value-passing processes and then we apply this theory to define a concrete model for our specific language. For the general theory 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. For this purpose we introduce the general notion of applicative signature (Σ, C) and that of (Σ, C)-terms where Σ is a set of operators and C a set of channel names. We also introduce the general class of applicative (Σ, C)-domains to model the semantics of the (Σ, C)-terms. These are a direct generalizations of the standard notion of signature, Σ, Σ–terms and Σ-domains originally introduced in [GTWW77] and used for instance in [Hen88a] to model a pure calculus. In the denotational interpretation of a language in terms of a (Σ, C)-domain the idea of the late semantic approach is made explicit; the outcome of an input action is modelled as a function which takes a value as an argument and returns an element of the model, i.e. a process, whereas the outcome of an output action is modelled by a pair consisting of the output value and the resulting process.

After having defined our general class of models we will modify the definition of evaluation mapping, i.e. the unique mapping from the process algebra into the domain known from the theory for pure processes. As we want to be able to reason about a subset of the process algebra, we extend the definition slightly. For this purpose we introduce the notion of *recursively closed subsets* of a process algebra. This extension of the definition allows us to reason about the compact elements of an algebraic *cpo* at the syntactic level. This enables us to take advantage of the notion of algebraicity when comparing the semantics defined by the model to other kinds of semantics such as behavioural or axiomatic semantics.

As the next step in the development of the general theory we apply the following general result for algebraic *cpo*s ([Hen88a]):

> Functions which are monotonic on the partial order consisting of 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. We may then use the standard result quoted above and take their unique continuous extension to be their definition on the whole domain.

By defining the operators this way, i.e. first as monotonic endofunctions on the partial order of compact elements and then extending them in a continuous way to the whole domain, we ensure that they *preserve compactness*. By this we mean that the result of applying an operator to a compact element is again a

compact element. From an intuitive point of view this is an important property; the compact elements represent the finitely computable elements of the domain so if we expect an operator *op* to be finitely computable, then applying it to something finitely computable should result in something finitely computable. Note that this property is not automatically satisfied in an applicative (Σ, C)-domain, or even a Σ-domain, as a continuous function does not necessarily map a compact element into a compact element. The following example illustrates this.

> Assume that $\langle D, \sqsubseteq \rangle$ is an algebraic *cpo* with the set of compact elements $Comp(D) \neq D$. Let $d_0 \in D \setminus Comp(D)$ and define the constant mapping $f_{d_0} : D \longrightarrow D$ by
>
> $$\forall d \in D. f_{d_0}(d) = d_0$$
>
> It is easy to show that f_{d_0} is continuous but that for any $d \in D$, $f_{d_0}(d) \notin Comp(D)$.

We complete the general theory by describing a procedure to construct the mentioned (Σ, C)-structure on a predefined algebraic *cpo*.

Next we define a concrete language, *Late-CCS* (CCS_L) by instantiating the general applicative signature (Σ, C). This language is a slight modification of the standard *CCS* where the syntax is basically the same as for the Polyadic π-calculus although we use a slightly different notation and only allow the transmission of simple values in communications. Then we define a concrete denotational model for CCS_L, the domain of *Applicative Communication Trees* (ACT), based on the general theory described above. It is an instantiation of the general class of (Σ, C)-domains, where Σ is instantiated with the operators of CCS_L. The definition of this model is motivated by the following models that have been studied in the literature.

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 the 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 [Smy78] 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 domain equation over a suitable class of domains. The process domain is then defined as the initial solution to this equation.

In [Abr91] 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 defined 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 defined his model in terms of the Plotkin Power Domain instead of the Smyth Power Domain. He added 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 interpreted the calculus $SCCS$ in the model and showed the full abstractness of this interpretation with respect to a bisimulation based preorder.

The model we define is basically the one presented in [MM79] where the modifications of Abramsky's are adopted. 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. Then we apply the general theory described above to define the operators over such a domain, i.e. by defining them as monotonic endofunctions on the *po* of compact elements and then extending them to continuous functions on the whole domain.

The definition of the denotational model supports in a natural way a system of equations and inference rules. We define such a proof system and prove its soundness and completeness with respect to the model. The ω-algebraicity of the model together with the fact that the operators preserve compactness enables us to reduce the proof of 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 [Hen88a] a semantic theory for process algebras describing concurrent languages with pure synchronization is given by means of Σ-domains. Adding values to the language calls for more complicated mathematical structures to describe the semantics. In this section we define a general class of mathematical structures to model process algebras with values which support the late semantic approach described in the introduction. For this purpose we extend the general syntax and introduce the general class of applicative signatures, (Σ, C) and that of (Σ, C)-terms where, as usual, Σ is a set of operators but C a set of channel names. We then define the general class of (Σ, C)-domains which is a direct generalization of the standard Σ-domains introduced in [GTWW77]. In fact the (Σ, C)-domains are only a slight modification of the Natural Interpretations introduced in [HP80] and used in [HI93]. Then we introduce the notion of *recursively closed* subsets of a process algebra.

Next, in §2.3, we show how we may turn an algebraic *cpo* into a (Σ, C)-domain by defining the operators on the compact elements, making sure that they are monotonic and then extending them to the whole domain. We also study the relationship between the evaluation mappings from our generic process language into two different (Σ, C)-preorders.

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.

The general syntax is based on predefined expression languages for value expressions and boolean expressions. Thus we assume some predefined syntactic 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 syntactic category, $BExp$, of boolean expressions, ranged over by be. $BExp$ should at least include a test for equality between the elements of Exp. From such a predicate a test for membership of a finite set can easily be derived. Value expressions are supposed to be equipped with a notion of substitution of an expression for a value variable, 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 preassume an infinite set of process names, PN, ranged over by P, Q, etc. The set of (Σ, C)-*terms* is now given as the triple

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

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

$$\begin{array}{ll} Proc_{(\Sigma,C)} : & p ::= op(\underline{p}), op \in \Sigma \mid c?.f \mid c!.\pi \mid \tau.p \mid be \rightarrow p, p' \\ Fun_{(\Sigma,C)} : & f ::= [x]p \\ Pairs_{(\Sigma,C)} : & \pi ::= (e, p) \end{array}$$

where we use the notation $\underline{p}$ to denote a vector of terms in $Proc_{(\Sigma,C)}$. If the process names in PN are added as primitives to the syntax for $T_{(\Sigma,C)}$, we write $T_{(\Sigma,C)}(PN)$ for the resulting triple of (Σ, C)-terms, 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 structure of this syntax is basically the same as suggested by Milner in [Mil91] although the notation is slightly different. The action of inputting on channel c is given by $c?$ whereas the action of outputting on that channel is given by $c!$. The *function terms* are of the form $[x]p$, where x is a data variable and p a process term. These correspond to the *abstractions* in the above mentioned reference. The input prefixing becomes $c?.[x]p$. The *pair terms* are of the form (e, p), where e is a data expression and p a process term. These correspond to the *concretions* in [Mil91]. The output prefixing becomes $c!.(e, p)$. We also assume that we have a set of operators, Σ, which is supposed to contain at least 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. We use the notation $be \longrightarrow p, p'$ to denote the standard conditional choice usually written as If be then p else p' .

Prefixing by $[x]$ binds the data variable x and the recursion construct is a binding construct for process names. A value variable, x, is free if it is not in the scope of a prefix, $[x]$, and a process name 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 and pair terms with no free occurrences of value variables by $CProc_{(\Sigma,C)}$, $CFun_{(\Sigma,C)}$ and $CPairs_{(\Sigma,C)}$ respectively. These will be referred to as processes, functions and pairs ranged over by cp, cf and $c\pi$. We assume a notion of substitution for both data variables and process names in terms defined in the usual way. For $f = [x]p$ and $v \in Val$ we use the convention $f(v) = ([x]p)(v) = p[v/x]$.

In the theory to follow we will make an extensive use of the fact that the value domain Val is countable. As Val is countable it may be written as $Val = \{v_1, v_2, v_3, \ldots, \}$. By defining $V_n = \{v_1, \ldots, v_n\}$ we get that $Val = \bigcup_n V_n$. In what follows V_n will have this meaning.

2.2 (Σ, C)-Orders and (Σ, C)-Domains

In this subsection we define the notion of applicative orders and applicative ordered (Σ,C)-algebras. We borrow the notation from [Hen88a] and use the abbreviations *pro* for preorder, *po* for partial order and *cpo* for complete partial order. We assume that the reader is familiar with basic domain theory and algebraic semantics. (See e.g. [Plo81, Hen88a] for details.)

Definition 2.1 Applicative Orders A pair $\langle \overline{A}, \sqsubseteq_A \rangle$ is an applicative *pro/po/cpo* if

$$\overline{A} = (A_{proc}, A_{fun}, A_{pair})$$

and

$$\sqsubseteq_{\overline{A}} = (\sqsubseteq_{A_{proc}}, \sqsubseteq_{A_{fun}}, \sqsubseteq_{A_{pair}})$$

are 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:

$$(v_1, p_1) \sqsubseteq_{A_{pair}} (v_2, p_2) \text{ if } v_1 = v_2 \text{ and } p_1 \sqsubseteq_{A_{proc}} p_2.$$

$\overline{A}$ 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.

□

Example 2.2 *Consider the domain* $\mathbf{2} = \{\bot, \top\}$ *with the standard ordering as* A_{proc}*, the po of compact elements of the domain* $[Val \longrightarrow \mathbf{2}]$ *as* A_{fun} *(i.e.* $A_{fun} = Val \longrightarrow_{fin} \mathbf{2} = \{f \in Val \longrightarrow \mathbf{2} \mid \{v \in Val | f(v) = \top\}$ *is finite* $\}$*) and* $A_{pair} = Val \times \mathbf{2}$*. This is an example of an applicative po which is not fully applicative. For instance the function* $f = \lambda v \in Val.\top$ *is not a compact element of* $[Val \longrightarrow \mathbf{2}]$ *and thus is not an element of* A_{fun}*.*

We often write a-pro/po/cpo as a shorthand for applicative pro/po/cpo.

Definition 2.3 **[(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 a-$pro/po/cpo$.
2. $\langle A_{proc}, \sqsubseteq_{A_{proc}}, \Sigma_{\overline{A}} \rangle$ is a $\Sigma - pro/po/cpo$ in the sense of [Hen88a].
3. $C_{\overline{A}} = C!_{\overline{A}} \cup C?_{\overline{A}}$ where:
   (a) $C!_{\overline{A}}$ is a set of monotonic/monotonic/continuous functions $c!._{\overline{A}} : A_{pair} \to A_{proc}$.
   (b) $C?_{\overline{A}}$ 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*. For an algebraic cpo A, we use $Comp(A)$ to denote the set of compact elements of A. □

Definition 2.4 A function $f : A_1 \longrightarrow A_2$, where $\langle A_1, \sqsubseteq_1 \rangle$ and $\langle A_2, \sqsubseteq_2 \rangle$ are algebraic *cpo*s, is said to be *compact* if it it maps compact elements of A_1 into compact elements of A_2, i.e. if $f(Comp(A_1)) \subseteq f(Comp(A_2))$. □

Next we extend the standard notion of homomorphisms for applicative orders.

Definition 2.5 A a-$pro/po/cpo$ homomorphism $\overline{h} : \langle \overline{A}, \sqsubseteq_{\overline{A}} \rangle \longrightarrow \langle \overline{B}, \sqsubseteq_{\overline{B}} \rangle$ is a triple 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 triple, $(h_{proc}, h_{fun}, h_{pair})$, where $h_{proc} : A_{proc} \longrightarrow B_{proc}$ is a $\Sigma - pro/po/cpo$ homomorphism in the sense of [Hen88a], $h_{fun} : A_{fun} \longrightarrow B_{fun}$ and $h_{pair} : A_{pair} \longrightarrow B_{pair}$ are $pro/po/cpo$ homomorphisms and satisfy:

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

□

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 $f_i : A_i \longrightarrow B_i$, $i = 1, 2, 3$. All the relations we use will be extended pointwise to vectors without further explanations.

Sometimes it is useful to be able to apply structural induction on a sublanguage of the full language defined by an a-signature, (Σ, C), and a set of process names, PN. In particular we want to be able to give recursive definitions on certain sublanguages. This motivates the following definition of a *recursively closed* subset of a language.

Definition 2.6 $\overline{S} = (S_{proc}, S_{fun}, S_{pair}) \subseteq T_{(\Sigma,C)}(PN)$ is said to be *recursively closed* if the following hold:

1. $p = op(p_1, \ldots, p_n) \in S_{proc}$ implies $p_i \in S_{proc}$ for $i = 1, \ldots, n$.
2. $c?.f \in S_{proc}$ implies $f \in S_{fun}$,
3. $c!.\pi \in S_{proc}$ implies $\pi \in S_{pair}$,
4. $be \longrightarrow p_1, p_2 \in S_{proc}$ implies $p_1, p_2 \in S_{proc}$,
5. $[x]p \in S_{proc}$ implies $p[v/x] \in S_{proc}$ for all $v \in Val$,
6. $(e, p) \in S_{proc}$ implies $p \in S_{proc}$.

In this case we write $\overline{S} \subseteq_{rec} T_{(\Sigma,C)}(PN)$. □

Note that if $\Sigma' \subseteq \Sigma$ and $C' \subseteq C$ then $T_{(\Sigma',C')}(PN) \subseteq_{rec} T_{(\Sigma,C)}(PN)$.

Definition 2.7 Let $\overline{S} \subseteq_{rec} T_{(\Sigma,C)}(PN)$, $\langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}} \rangle$ be an applicative $(\Sigma, C) - pro$ and $PEnv_{\overline{X}}$ be the set of process environments $\rho : PN \longrightarrow X_{proc}$. A function

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

is an *evaluation* function if it satisfies:

$$\begin{aligned}
X_{proc}[\![op(\underline{p})]\!]\rho &= op_{\overline{X}}(X_{proc}[\![\underline{p}]\!]\rho),\ op \in \Sigma \\
X_{proc}[\![c?.f]\!]\rho &= c?_{\overline{X}}.X_{fun}[\![f]\!]\rho \\
X_{proc}[\![c!.\pi]\!]\rho &= c!_{\overline{X}}.X_{pair}[\![\pi]\!]\rho \\
X_{proc}[\![\ be \rightarrow p_1, p_2]\!]\rho &= \begin{cases} X_{proc}[\![p_1]\!]\rho & \text{if } [\![be]\!] = T \\ X_{proc}[\![p_2]\!]\rho & \text{if } [\![be]\!] = F \end{cases} \\
X_{fun}[\![[x]p]\!]\rho &= \lambda v.X_{proc}[\![p[v/x]]\!]\rho \\
X_{pair}[\![(e,p)]\!]\rho &= ([\![e]\!], X_{proc}[\![p]\!]\rho) \\
X_{proc}[\![P]\!]\rho &= \rho(P)
\end{aligned}$$

If $\overline{X}$ is a *cpo* then, following the standard practice, we may define

$$X_{proc}[\![rec\,P.p]\!]\rho = Y\ \lambda d.X_{proc}[\![p]\!]\rho[d/P]$$

where Y is the least fixed point operator. □

For closed terms the environments do not have any influence on the definition. For process name free terms a mapping $\overline{X}[\![ct]\!] = \overline{X}[\![ct]\!]\rho$ may be derived from the above definition omitting the last clause of the definition and the occurrence of ρ in the others.

Now we show that recursively closed subsets of $T_{(\Sigma,C)}(PN)$ have at most one interpretation in an a-(Σ, C)-pro. This is the subject of the next theorem.

Theorem 2.8 *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.*

Proof May be proved by structural induction and is left to the reader. □

Note that if $\overline{X}$ is not fully applicative then a function term of the form $[x].p$ may fail to have an interpretation in $\overline{X}$. For instance if p is a term denoting $\top$ in the applicative po considered in Example 2.2 then the function term $[x]p$ fails to have an interpretation in A_{fun}.

The following result turns out to be useful in the next section.

Corollary 2.9 *Assume that*

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

that $\langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}} \rangle$ *and* $\langle \overline{Y}, \prec_{\overline{Y}}, \Sigma_{\overline{Y}}, C_{\overline{Y}} \rangle$ *are* (Σ, C)*-pros and that*

$$\overline{\psi} : \langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}} \rangle \longrightarrow \langle \overline{Y}, \prec_{\overline{Y}}, \Sigma_{\overline{Y}}, C_{\overline{Y}} \rangle$$

is a (Σ, C)*-pro homomorphism. If* $\overline{X}[\![_]\!] : \overline{S} \longrightarrow \langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}} \rangle$ *and* $\overline{Y}[\![_]\!] : \overline{S} \longrightarrow \langle \overline{Y}, \prec_{\overline{Y}}, \Sigma_{\overline{Y}}, C_{\overline{Y}} \rangle$ *are evaluation mappings, then* $\overline{Y}[\![_]\!] = \overline{\psi} \circ \overline{X}[\![_]\!]$.

Proof It is easy to check that the mapping $\overline{Y}[\![_]\!]$ defined by $\overline{Y}[\![_]\!] = \overline{\psi} \circ X[\![_]\!]$ is an evaluation mapping from $\overline{S}$ to $\langle \overline{Y}, \prec_{\overline{Y}}, \Sigma_{\overline{Y}}, C_{\overline{Y}} \rangle$. By Theorem 2.8 such an evaluation mapping is unique and the equality follows. □

2.3 Properties Derived From the Compact Elements

In this subsection we will describe how we can take advantage of the algebraicity of an applicative (Σ, C)-domain, $\langle \overline{A}, \sqsubseteq_{\overline{A}}, \Sigma_{\overline{A}}, C_{\overline{A}} \rangle$ to obtain a full description of certain properties of the domain from the knowledge of the same properties only on the partial order consisting of the compact elements of the model. In fact we do more than that: we take an applicative ω-algebraic *cpo*, $\langle \overline{B}, \sqsubseteq_{\overline{B}} \rangle$, and show how it may be turned into an applicative (Σ, C)-domain by defining the interpretation of the operators in Σ and C on an applicative *preorder* that represents the partial order of the compact elements of the model $\overline{B}$. (By a

representation of a partial order we mean a preorder whose induced partial order obtained by factoring out the preorder is isomorphic to the original one.) If the operators are monotonic they induce in a unique way continuous operators defined on the whole of $\overline{B}$. The following standard theorem (see e.g. [Hen88a]) plays an important role in this connection.

Theorem 2.10 *Let A and B be cpos. Assume that A is algebraic and let $f : Comp(A) \longrightarrow B$ be monotonic. Then there exists a unique continuous extension of f, $\tilde{f} : A \longrightarrow B$.*

The main result of this subsection is stated in the following theorem where

$$\overline{inc} : \langle Comp(\overline{B}), \sqsubseteq_{Comp(\overline{B})} \rangle \longrightarrow \langle \overline{B}, \sqsubseteq_{\overline{B}} \rangle$$

is the inclusion mapping and $[_]_\sim : \overline{X} \longrightarrow \overline{X}/\sim$ is the quotient mapping.

Theorem 2.11 *Assume that $\langle \overline{X}, \prec_{\overline{X}}, \Sigma_{\overline{X}}, C_{\overline{X}} \rangle$ is a $(\Sigma, C)-pro$ with the induced partial order $\langle \overline{X}/\sim, \preceq_{\overline{X}/\sim} \rangle$ where $\sim = \prec_{\overline{X}} \cap \prec_{\overline{X}}^{-1}$. Further assume that $\langle \overline{B}, \sqsubseteq_{\overline{B}} \rangle$ is an applicative algebraic cpo whose po of compact elements, $\langle Comp(\overline{B}), \sqsubseteq_{Comp(\overline{B})} \rangle$, is isomorphic to $\langle \overline{X}/\sim, \preceq_{\overline{X}/\sim} \rangle$ under the isomorphism*

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

Then the following holds:

1. *There exists a unique (Σ, C)-structure, $(\Sigma_{\overline{B}}, C_{\overline{B}})$ which extends $\langle \overline{B}, \sqsubseteq_{\overline{B}} \rangle$ to a (Σ, C)-domain and extends $\overline{\psi} = \overline{inc} \circ \overline{\phi} \circ [_]_\sim$ to a (Σ, C)-homomorphism. The structure $(\Sigma_{\overline{B}}, C_{\overline{B}})$ is compact in the sense of Definition 2.4.*

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

$$\overline{B}[\![_]\!]|_{\overline{S}} = \overline{\psi} \circ \overline{X}[\![_]\!]$$

where $\overline{B}[\![_]\!]|_{\overline{S}}$ means the restriction of the function $\overline{B}[\![_]\!]$ to $\overline{S}$.

Proof

1. Existence: Let $\overline{B}_1 = Comp(\overline{B})$. We note that $inc(c) = c$ for all $c \in \overline{B}_1$. By assumption $\overline{\phi} \circ [_]_\sim : \overline{X} \longrightarrow \overline{B}_1$ is monotonic and surjective. In particular any element of $\overline{B}_1$ may be written as $\overline{\phi}([x]_\sim)$ for some $x \in \overline{X}$. Now let $op \in \Sigma$. We define the operator $op_{\overline{B}_1}$ by

$$op_{\overline{B}_1}(c) = op_{\overline{B}_1}(\phi_{proc}([x]_\sim)) = \phi_{proc}([op_{\overline{X}}(x)]_\sim)$$

for all $c = \overline{\phi}([x]_\sim) \in Comp(\overline{B})$. It is easy to check that $op_{\overline{B}_1}$ is well defined and monotonic. Then we take $op_{\overline{B}}$ to be the unique continuous extension to $\overline{B}$ given by Theorem 2.10. We define $C_{\overline{B}}$ in a similar way. Thus we obtain well defined and continuous operators and prefixings on $\langle \overline{B}, \sqsubseteq_{\overline{B}} \rangle$. The compactness also follows directly from the

definition. It remains to prove that the structure $(\Sigma_{\overline{B}}, C_{\overline{B}})$ extends $\overline{\psi}$ to a (Σ, C)-homomorphism. So take $x \in X_{proc}$. By definition of $op_{\overline{B}_1}$ we get

$$\psi_{proc}(op_{\overline{X}}(x)) = inc(\phi_{proc}([op_{\overline{X}}(x)]_\sim)) = inc(op_{\overline{B}_1}(\phi_{proc}([x]_\sim))) = op_{\overline{B}_1}(\phi_{proc}([x]_\sim)) = op_{\overline{B}}(\phi_{proc}([x]_\sim)) = op_{\overline{B}}(\psi_{proc}(x)).$$

Uniqueness: Assume $(\Sigma'_{\overline{B}}, C'_{\overline{B}})$ is a structure that extends $\overline{\psi}$ as described in the theorem. We have to show that $\Sigma'_{\overline{B}} = \Sigma_{\overline{B}}$ and $C'_{\overline{B}} = C_{\overline{B}}$. We will only show the first equality as the proof for the other one is similar and is left to the reader. So let $op'_{\overline{B}} \in \Sigma'_{\overline{B}}$ be the operator named by op. We will show that $op'_{\overline{B}} = op_{\overline{B}}$. By assumption both $op_{\overline{B}}$ and $op'_{\overline{B}}$ are continuous, so by Theorem 2.10 it is sufficient to prove they coincide on the compact elements of the domain. Let $c \in Comp(\overline{B})$. It is sufficient to prove that $op'_{\overline{B}}(c) = op_{\overline{B}}(c)$. So, as $\overline{\psi} = \overline{\phi} \circ [_] : \overline{X} \longrightarrow Comp(\overline{B})$ is onto, we have that there is an $x \in X_{proc}$ such that $c = \psi_{proc}(x)$. Thus, as $(\Sigma'_{\overline{B}}, C'_{\overline{B}})$ extends $\overline{\psi}$ to a (Σ, C)-homomorphism, the definition of $op_{\overline{B}}$ gives

$$op'_{\overline{B}}(c) = op'_{\overline{B}}(\psi_{proc}(x)) = \psi_{proc}(op_{\overline{X}}(x)) = \phi_{proc}([op_{\overline{X}}(x)]_\sim) = op_{\overline{B}}(c).$$

This proves the uniqueness.

2. It is easy to check that $\overline{B}[\![_]\!]|_{\overline{S}}$ is an evaluation mapping on $\overline{S}$ and the result follows directly from Corollary 2.9.

□

3 Late CCS and Its Denotational Semantics

In this section we will give a concrete language, *Late* CCS, (CCS_L) by instantiating the applicative signature (Σ, C). Furthermore we will define a concrete (Σ, C)-domain to give a denotational semantics for this language.

3.1 The Language

The language CCS_L 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 $CCS_L(PN) = (CCS_L^{proc}(PN), CCS_L^{fun}(PN), CCS_L^{pair}(PN))$ 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 $_\backslash c, c \in C$ and $_[R]$ where

$$
\begin{array}{lll}
CCS_L^{proc}(PN) := & NIL \mid \Omega \mid p[R] \mid p \setminus c \mid p + p \mid p|p \mid c?.f \mid c!.\pi \mid \tau.p \mid \\
& be \rightarrow p, p \mid P \mid recP.p \\
CCS_L^{fun}(PN): & f ::= [x]p \\
CCS_L^{pair}(PN): & \pi ::= (e, p)
\end{array}
$$

Figure 1: The Syntax for CCS_L

R is a finite permutation of the channel names (i.e. $R : C \longrightarrow C$ is constant on all but finitely many channels in C) and the binary operators $+$ and $|$. For the motivation of these operators we refer to the standard theory of CCS in [Mil89]. For the sake of clarity the syntax for $CCS_L(PN)$ is given in Figure 1. We let $CCS_L = (CCS_L^{proc}, CCS_L^{fun}, CCS_L^{pair})$ denote the closed terms in $CCS_L(PN)$.

3.2 A Domain Equation for Applicative Communication Trees

In this section we will construct a (Σ, C)-domain which will be used to give the denotational semantics for our language, CCS_L. The model we define is basically the model of [MM79] where the modifications of Abramsky's, reported in [Abr91] and described in the introduction, are adopted. Thus we define a model for 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 the Plotkin Power Domain to the one used in [Abr91]. The representation we use is the one due to Smyth, [Smy78] and will be described below. In the definition of the domain we use the following operations on cpos:

Cartesian product $\times$: ([Plo81], §2 and §6) Let $\langle A, \sqsubseteq_A \rangle$ and $\langle A', \sqsubseteq_{A'} \rangle$ be two *po*s. We define the partial order $\sqsubseteq_{A \times A'}$ on $A \times A'$ by:

$$(a, a') \sqsubseteq_{A \times A'} (b, b') \text{ if } a \sqsubseteq_A b \text{ and } a' \sqsubseteq_{A'} b'$$

This construction extends to any number of *po*s. It preserves completeness and algebraicity. Countable products preserve ω-algebraicity. If A and A' are algebraic *cpo*s, the set of compact elements can be obtained from the compact elements of A and A' by $Comp(A \times A') = Comp(A) \times Comp(A')$.

Separated Sum $\sum_{i \in I}$: ([Abr87], §3, [Plo81], §3 and §6) Let I be a countable index set and $\{A_i\}_{i \in I}$ be a family of I–indexed *po*s. The separated sum $\langle \sum_{i \in I} A_i, \sqsubseteq_{\sum_{i \in I} A_i} \rangle$ is defined as follows:

$$\sum_{i \in I} A_i = \{\bot\} \cup (\bigcup \{\{i\} \times A_i | i \in I\})$$

$$x \sqsubseteq_{\sum_{i \in I} A_i} y \text{ if } x = \bot \text{ or if for some } i,\ x = \langle i, a \rangle,\ y = \langle i, a' \rangle \text{ and } a \sqsubseteq_{A_i} a'$$

where we write $\langle i, a \rangle$ for the elements of the disjoint union and $\bot$ for the bottom element of the separated sum. The construction preserves

completeness, algebraicity and ω-algebraicity. If each A_i is an algebraic *cpo*, the set of compact elements of $\langle \sum_{i\in I} A_i, \sqsubseteq_{\sum_{i\in I} A_i} \rangle$ is given by

$$Comp(\sum_{i\in I} A_i) = \{\bot\} \cup (\bigcup \{\{i\} \times Comp(A_i) | i \in I\}$$

Function Space from a fixed set, S, F_S: ([Plo81], §3) Let S be a fixed countable set. For a *po* $\langle A, \sqsubseteq_A \rangle$ we define $F_S(A) = S \longrightarrow A$, the set of all functions from S to A, with the pointwise ordering, $\sqsubseteq_{F_S(A)}$, as follows:

$$f \sqsubseteq_{F_S(A)} g \text{ if } \forall s \in S. f(s) \sqsubseteq_A g(s).$$

This construction preserves completeness, algebraicity and ω-algebraicity. The compact elements of $F_S(A)$ can be obtained from those of A by $Comp(F_S(A)) = F_S^{fin}(Comp(A))$ where $F_S^{fin}(B) = \{f \in S \longrightarrow B | \{s \in S | f(s) \neq \bot\} \text{ is finite}\}$. Note that the constructions $\sum_{i\in I}$ and $F_S(A)$ may just as well be defined for non-countable sets I and S but then they do not preserve ω-algebraicity in general.

Completion by Ideals: ([Hen88a], §3.3,[Win85]) There is a standard way of extending a *preorder* with a least element to an algebraic *cpo*, often called *completion by ideals*. Let $\langle A, \sqsubseteq_A \rangle$ be a preorder. A set $X \subseteq A$ is *downwards closed* if whenever $x \in X$ and $y \sqsubseteq_A x$ then $y \in X$. An *ideal* in A is a non-empty, directed and downwards closed subset of A. Let $\mathcal{I}(A)$ denote the set of all ideals in A. If A has a least element then $\langle \mathcal{I}(A), \subseteq \rangle$ is an algebraic *cpo*. The compact elements of $\mathcal{I}(A)$ are $Comp(\mathcal{I}(A)) = \{\downarrow a | a \in A\}$ where $\downarrow a = \{x | x \sqsubseteq_A a\}$. $\mathcal{I}(A)$ is the unique algebraic *cpo* (up to isomorphism) whose partial order of compact elements is isomorphic to the kernel of $\langle A, \sqsubseteq_A \rangle$, i.e. $\langle A/=_A, \sqsubseteq_{A/=_A} \rangle$ where $=_A$ is the equivalence induced by $\sqsubseteq_A$. This is referred to as the ideal completion of $\langle A, \sqsubseteq_A \rangle$. Note that if $A/=_A$ is countable then $\mathcal{I}(A)$ is ω-algebraic.

The Plotkin Power Domain: ([Win85]) We give a construction of the *Plotkin Power Domain* [Plo76] due to Smyth, [Smy78], and described in [Win85]. Let $\langle A, \sqsubseteq_A \rangle$ be an ω-algebraic *cpo* and $M[A]$ the family of finite, non-empty sets of compact elements of A. The *Egli-Milner order* on $M[A]$ is defined by:

$$\text{For } X, Y \in M[A],\ X \sqsubseteq_{EM} Y \text{ iff } \forall x \in X \exists y \in Y. x \sqsubseteq_A y \text{ and}$$

$$\forall y \in Y \exists x \in X. x \sqsubseteq_A y.$$

The Plotkin Power Domain of $\langle A, \sqsubseteq_A \rangle$, $\langle P[A], \sqsubseteq_{P[A]} \rangle$ is the ideal completion of the preorder $\langle M[A], \sqsubseteq_{EM} \rangle$. From above-mentioned results, we know that $\langle P[A], \sqsubseteq_{P[A]} \rangle$ is an ω-algebraic *cpo* and $Comp(P[A]) = M[A]/=_{EM}$ (up to isomorphism).

In the definition to follow we shall use Abramsky's modification of the Plotkin Power Domain, i.e we add the empty set to the domain in such a way that it is only related to itself and the least element of the domain in the obvious way under the extended Egli-Milner order. This may be described as follows:

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:

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

All the constructions on pos described above may be turned into covariant continuous functors in the category $\mathbf{CPO}^E$, the category of cpos with embeddings, in a straightforward way. For the details we refer to [Plo81]. Now the standard theory in [Plo81] ensures that the following definition is meaningful.

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

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

where

- $D_{c?} = F_{Val}(D) = Val \to D$ (as defined on page 14),
- $D_{c!} = Val \times D$ 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. $\sqsubseteq_{\overline{ACT}}$ is the applicative partial order induced by $\sqsubseteq_{ACT_{proc}} = \sqsubseteq_{ACT}$. Defined in this way $\langle \overline{ACT}, \sqsubseteq_{\overline{ACT}} \rangle$ is a fully applicative ω-algebraic *cpo* which we refer to as ACT. Also we let $\sqsubseteq$ denote $\sqsubseteq_{\overline{ACT}}$. □

From the general theory in [Plo81] we get a representation of the compact elements by unfolding the recursive definition of ACT. Thus we define

$$COMP = \bigcup_{n=0}^{\infty} COMP^n$$

where

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

and

$$COMP^{n+1} = M[\sum_{e \in Act} (COMP^n)_e] \cup \{\emptyset\}$$

where $(COMP^n)_{c?} = F_{Val}(COMP^n)$, $(COMP^n)_{c!} = Val \times COMP^n$ 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 $COMP$ this way and ordering it by $\sqsubseteq^0_{EM}$, the Egli-Milner preorder over $COMP$ extended like in (1) above, gives a representation of the compact elements of the ω-algebraic cpo ACT. This means that the kernel of the preorder is isomorphic to the partial order of compact elements of the domain ACT, $\langle Comp(ACT), \sqsubseteq_{Comp(ACT)}\rangle$. (For the sake of simplicity we assume that the kernel is *equal* to $Comp(ACT)$.) This suggests an inductive definition to describe the set $COMP$ and the preorder on $COMP$. Thus we define the set K and the preorder $\prec$ on it inductively and prove that $\langle K, \prec\rangle$ is equal to $\langle COMP, \sqsubseteq^0_{EM}\rangle$.

Definition 3.2 We define K as the least set which satisfies:

1. $\emptyset \in K$
2. $\{\bot\} \in K$
3. $c \in C, V \subseteq_{fin} Val$ 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. $c \in C, v \in Val, k \in K$ 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. $\{\bot\} \prec \emptyset$
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) $\forall a. \bot \leq a$
   (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 $\langle K, \prec\rangle = \langle COMP, \sqsubseteq^0_{EM}\rangle$.

Proof First we prove that $K = COMP$. That $K \subseteq COMP$ can be proved by showing that $COMP$ is closed under $1. - 6.$ in the definition of K and then use the fact that K is the least set with this property.

To prove the opposite inclusion, it is sufficient to show that, for every n, $COMP^n \subseteq K$. The details are left to the reader.

Then we prove that the preorder $\prec$ coincides with the extended Egli-Milner preorder on K. We have to prove the two following cases:

$\prec \subseteq \sqsubseteq^0_{EM}$: It is sufficient to prove that $\sqsubseteq^0_{EM}$ satisfies the definition of $\prec$; as $\prec$ is the least preorder which satisfies this definition the inclusion follows. The details are straightforward and are left to the reader.

$\prec \supseteq \sqsubseteq^0_{EM}$: To prove this case we first define the depth of the elements of K as follows:

1. $d(\emptyset) = d(\{\bot\}) = 0$
2. $d(\{\langle \mu, k\rangle\}) = 1 + d(k)$
3. $d(\{a_1, \ldots, a_n\}) = max\{d(\{a_i\})|i \leq n\}$
4. $d(f) = max\{d(f(v))|v \in Val\}$ (Recall that $\{f(v)|v \in Val\}$ is a finite set as f yields $\bot$ on all but finitely many values in Val.)
5. $d(e, k) = d(k)$

Now we proceed as follows: We will prove by induction on $d(k)$ that

$$k \sqsubseteq^0_{EM} k' \Rightarrow k \prec k'$$

So assume $k \sqsubseteq^0_{EM} k'$.

base $d(k) = 0$:

$k = \emptyset$: Then $k' = \emptyset$ and $k \prec k'$.

$k = \{\bot\}$: $k \prec k'$ is obvious in this case.

step $d(k) = n + 1$: Now $k \neq \emptyset$ so we may assume $a \in k$. Then by definition of $\sqsubseteq^0_{EM}$, $a \sqsubseteq_{P^o[COMP]} b$ for some $b \in k'$. We will prove that $a \leq b$. We have the four different cases: $a = \bot$, $a = \langle c?, f\rangle, a = \langle c!, (v, k_1)\rangle$ and $a = \langle \tau, k_2\rangle$. The first case is obvious. We only prove the statement for the second case as the proof for the remaining two is similar. So assume $a = \langle c?, f\rangle$. Then $b = \langle c?, g\rangle$, where $f(v) \sqsubseteq^0_{EM} g(v)$ for all $v \in Val$. Now the induction applies and we may conclude that $f(v) \prec g(v)$ for all $v \in Val$. From this we get that $a \leq b$ as wanted.

Next assume $b \in k'$, then again by definition of $\sqsubseteq^0_{EM}$ there is an $a \in k$ such that $a \sqsubseteq^0_{EM} b$. In the same manner as before we may conclude that $a \leq b$ which completes the proof of this inclusion.

□

Definition 3.4 We define $\langle \overline{K}, \prec_{\overline{K}}\rangle$ by letting $K_{proc} = K$, $K_{fun} = F^{fin}_{Val}(K)$ (where $F^{fin}_{Val}(K)$ is defined as on page 14) and $K_{pair} = Val \times K$ and by defining $\prec_{\overline{K}}$ as the preorder induced by $\prec_{proc} = \prec$. □

3.3 Definition of the Operators in the Model

In this subsection we will define the operators in Σ_{ACT} and prove their continuity. In the definitions we take advantage of Theorem 2.11. Thus we only have to define the operators on the applicative preorder $\langle \overline{K}, \prec_{\overline{K}} \rangle$ and make sure that they are monotonic.

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

Constants:

$$\begin{aligned} NIL_{\overline{K}} &= \emptyset \\ \Omega_{\overline{K}} &= \{\bot\} \end{aligned}$$

Prefixing:

$$\begin{aligned} c?_{\overline{K}}_ &= \lambda f.\{\langle c?, f\rangle\} \\ c!_{\overline{K}}_ &= \lambda(v,k).\{\langle c!,(v,k)\rangle\} \\ \tau_{\overline{K}}_ &= \lambda k.\{\langle \tau, k\rangle\} \end{aligned}$$

Nondeterminism:

$$+_{\overline{K}} = \cup$$

Restriction:

$$_\backslash c_{\overline{K}} = F_c$$

where $F_c : K_{proc} \to K_{proc}$ is defined by

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

Renaming:

$$_[R]_{\overline{K}} = G_R$$

where $G_R : K_{proc} \to K_{proc}$ is defined by

$$\begin{aligned} G_R\{\bot\} &= \{\bot\} \\ G_R\emptyset &= \emptyset \end{aligned}$$

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

Parallel Composition:

$$
_ |_{\overline{K}} _ = F
$$

where $F = int \cup comm \cup div$ where $int = int_{in} \cup int_{out} \cup int_\tau$ and

$$
\begin{aligned}
int_{in}(x,y) &= \{\langle c_x?, \lambda v.F(f_x(v), y)\rangle | \langle c_x?, f_x\rangle \in x\} \\
&\cup \{\langle c_y?, \lambda v.F(x, f_y(v))\rangle | \langle c_y?, f_y\rangle \in y\} \\
int_{out}(x,y) &= \{\langle c_x!, (v, F(x', y)\rangle | \langle c_x!, (v, x')\rangle \in x\} \\
&\cup \{\langle c_y!, (v, F(x, y')\rangle | \langle c_y!, (v, y')\rangle \in y\} \\
int_\tau &= \{\langle \tau, F(x', y)\rangle | \langle \tau, x'\rangle \in x\} \\
&\cup \{\langle \tau, F(x, y')\rangle | \langle \tau, y'\rangle \in y\}
\end{aligned}
$$

$$
\begin{aligned}
comm(x,y) &= \{\langle \tau, F(f(v), y')\rangle \mid \exists c, v. \langle c?, f\rangle \in x \text{ and } \langle c!, (v, y')\rangle \in y\} \\
&\cup \{\langle \tau, F(x', g(v))\rangle \mid \exists c, v. \langle c?, g\rangle \in y \text{ and } \langle c!, (v, x')\rangle \in x\}
\end{aligned}
$$

and

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

□

The reader may notice the close connection between the definition of the parallel operator and the interleaving law presented later in the paper. We have the following result:

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

Proof We leave it to the reader to check that the operators defined by Definition 3.5 are well-defined. The monotonicity of the operators $NIL_{\overline{K}}$, $\Omega_{\overline{K}}$, $c?_{\overline{K}}$, $c!_{\overline{K}}$, $\tau_{\overline{K}}$ and $+_{\overline{K}}$ is obvious. To prove the monotonicity of the remaining operators we use the depth, $d(_)$, of the elements of K defined in the proof of Proposition 3.3. To prove the monotonicity of the restriction and the renaming operators we prove by induction on $d(k)$ that

$$
k \prec k' \text{ implies } F_c(k) \prec F_c(k')
$$

and

$$
k \prec k' \text{ implies } G_R(k) \prec G_R(k')
$$

To prove the monotonicity of the parallel operator with respect to the induced ordering on $K \times K$, we extend d to $K \times K$ by

$$d(k_1, k_2) = d(k_1) + d(k_2)$$

Now we may prove that

$$(k_1, k_2) \prec (k_1', k_2') \text{ implies } F(k_1, k_2) \prec F(k_1', k_2')$$

by induction on $d(k_1, k_2)$. We leave the straightforward details of the proof to the reader. □

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) structure induced by $\langle \overline{K}, \prec_{\overline{K}}, \Sigma_{\overline{K}}, C_{\overline{K}} \rangle$, is a fully applicative (Σ, C)-domain. The operators in $(\Sigma_{\overline{ACT}}, C_{\overline{ACT}})$ and $C_{\overline{ACT}}$ are compact.*

3.4 Syntactically Compact Elements

We will now show that the compact elements of the model $\overline{ACT}$ may be denoted in our syntax by a recursively closed subset of the whole language. For this purpose we introduce the so-called *syntactically compact Terms*, $CoTerms(PN) = (CoProc(PN), CoFun(PN), CoPair(PN))$.

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

Notation 3.8 *Let $\underline{w}_n = (w_1, \ldots, w_n)$ and $\underline{p}_n = (p_1, \ldots, p_n)$ be vectors of values and processes respectively. We write $x : \underline{w}_n \longrightarrow \underline{p}_n$ for $x = w_1 \longrightarrow p_1, (x = w_2 \longrightarrow p_2, (\ldots x = w_n \longrightarrow p_n, \Omega)\ldots)$. (Intuitively $x : \underline{w_n} \longrightarrow \underline{p_n}$ stands for the function that maps w_i to p_i for $i = 1, \ldots, n$ and all the other values $w \in Val$ into Ω.) Further we let $\{\underline{w_n}\} = \{w_i | \underline{w_n} = (w_1, \ldots, w_n)\}$ and similarly for $\{\underline{p_n}\}$.*

Definition 3.9 [Syntactically Compact Terms] The set of syntactically compact terms is the triple

$$CoTerms(PN) = (CoProc(PN), CoFun(PN), CoPairs(PN))$$

where the sets $CoProc(PN), CoFun(PN)$ and $CoPairs(PN)$ are the least sets satisfying:

1. $NIL, \Omega \in CoProc(PN)$ and $P \in CoProc(PN)$ for all $P \in PN$
2. $\overline{p} \in CoProc(PN)$ implies $op(\overline{p}) \in CoProc(PN)$, $op = |, +, _\backslash, _[R], \tau._$
3. $\pi \in CoPair(PN)$, $c \in C$ implies $c!\pi \in CoProc(PN)$

4. $f \in CoFun(PN)$ and $c \in CR$ implies $c?f \in CoProc(PN)$

5. $p \in CoProc(PN)$ and $e \in Exp$ implies $(e,p) \in CoPairs(PN)$

6. $\{\underline{p_n}\} \subseteq CoProc(PN)$, $\{\underline{w_n}\} \subseteq Val$ and $x \in Var$ implies $[x].x : \underline{w_n} \longrightarrow \underline{p_n} \in CoFun(PN)$.

We use the convention $CoTerms = CoTerms(\emptyset)$, $CoProc = CoProc(\emptyset)$, etc. and let them be ranged over by Cot, Cop, etc. We say that a term is *compact* if it belongs to $CoTerms(PN)$.

□

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

Theorem 3.10

1. *$\overline{ACT}[\![_]\!] : CCS_L \longrightarrow \overline{ACT}$ and $\overline{K}[\![_]\!] : CoTerms \longrightarrow \overline{K}$ are well-defined.*

2. *$\overline{ACT}[\![_]\!] \mid_{CoTerms} = inc \circ [_]_{\approx} \circ \overline{K}[\![_]\!]$ where*

   - *$f|_A$ means the restriction of the function f to the set A*
   - *$[_]_{\approx}$ is the quotient mapping with respect to the preorder $\prec$ and*
   - *$inc : \overline{K}/\approx = Comp(\overline{ACT}) \longrightarrow \overline{ACT}$ is the inclusion mapping.*

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

4. *For all $Cot_1, Cot_2 \in CoTerms$ $\overline{ACT}[\![Cot_1]\!] \sqsubseteq \overline{ACT}[\![Cot_2]\!]$ if and only if $\overline{K}[\![Cot_1]\!] \prec \overline{K}[\![Cot_2]\!]$*

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

Proof

1. As $\overline{ACT}$ is a fully applicative (Σ, C)-cpo then, by Theorem 2.8,

$$\overline{ACT}[\![_]\!] : CCS_L \longrightarrow \overline{ACT}$$

   is well-defined. The existence of $\overline{K}[\![_]\!]$ follows by a simple structural induction on $CoTerms$.

2. As $CoTerms$ is a recursively closed subset of CCS_L, we may conclude that
   $\overline{ACT}[\![_]\!]|_{CoTerms}$ is an evaluation mapping on $CoTerms$. By construction of $\langle \overline{ACT}, \sqsubseteq_{\overline{ACT}}, \Sigma_{\overline{ACT}}, C_{\overline{ACT}} \rangle$, $inc \circ [_]_{\approx} : \overline{K} \longrightarrow \overline{ACT}$ is a (Σ, C)-po homomorphism. It then follows from Theorem 2.11 that $\overline{ACT}[\![_]\!] \mid_{CoTerms} = inc \circ [_]_{\approx} \circ \overline{K}[\![_]\!]$.

3. This part follows immediately from part 2.

4. Follows from part 2. as well.

5. We start by proving that for any $k \in \overline{K}$ there is a $Cot \in CoTerms$ such that $\overline{K}[\![Cot]\!] = k$. First we prove the result for the set K which by definition equals K_{proc}. This may be proved by induction on the definition of K. Then we may easily extend the proof to K_{fun} and K_{pair}.

   Next assume that $c \in Co(\overline{ACT})$. Then $c = [k]_{\approx}$ for some $k \in \overline{K}$. From what we proved above we get that $\overline{K}[\![Cot]\!] = k$ for some $Cot \in CoTerms$. From 1. we get

$$\overline{ACT}[\![Cot]\!] = [\overline{K}[\![Cot]\!]]_{\approx} = [k]_{\approx}.$$

   This completes the proof.

□

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. As an example the equations

$$\begin{aligned} X + (Y + Z) &= (X + Y) + Z \\ X + Y &= Y + X \\ X + X &= X \end{aligned}$$

reflect the fact that the elements of K are defined as sets and $+$ as set union. 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

$$(res\,in) \quad (a?.[x]X) \setminus c = \begin{cases} a?.[x](X \setminus c) & \text{if } c \neq a \\ NIL & \text{otherwise} \end{cases}$$

$$(res\,out) \quad (a!.(e, X)) \setminus c = \begin{cases} a!.(e, X \setminus c) & \text{if } c \neq a \\ NIL & \text{otherwise} \end{cases}$$

$$(ren\,in) \quad (a?.[x]X)[R] = R(a)?.[x](X[R])$$

$$(ren\,out) \quad (a!.(e, X))[R] = R(a)!.(e, X[R])$$

and the rules

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

$$(pair) \quad \frac{[\![e_1]\!] = [\![e_2]\!], p \sqsubseteq q}{(e_1, p) \sqsubseteq (e_2, q)}$$

that allow us to prove inequalities over function terms and pairs. The extended system E_{rec} is then obtained by adding to E three new rules to take care of recursion. These rules are all fairly standard and will not be explained here (see e.g. [Hen88a]). The new rules introduced are

$$(rec) \quad recP.p = p[recP.p/P]$$

and

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

where $p^{(n)}$, the syntactically compact approximations of a process term p, are defined in Definition 4.1. Note here that the approximations that occur in the ω-rule are syntactically compact as the number of values in the approximations is finite just as the depth of the approximation is finite. This enables us to take advantage of the algebraicity of the model when proving the completeness of the proof system. It is possible that the weaker version of the ω-rule with the more standard syntactically finite approximations, i.e. without restrictions on the values, would yield a complete system as well but then a more complicated proof technique would be needed. 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 a summand of X or Y on the left hand side. 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. Now we define the syntactically compact approximations used in the ω-rule of the proof system.

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

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. $(be \longrightarrow p, q)^{(n+1)} = \begin{cases} p^{(n+1)} & \text{if } [\![be]\!] = T \\ q^{(n+1)} & \text{if } [\![be]\!] = F \end{cases}$

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

III. $((e,p))^{(n+1)} = \begin{cases} ([\![e]\!], p^{(n+1)}) & \text{if } [\![e]\!] \in V_{n+1} \\ ([\![e]\!], \Omega) & \text{otherwise} \end{cases}$

□

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

Figure 2: Equations

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 3: Interleaving Law

(ref) $\quad p \sqsubseteq p$

$(trans)$ $\quad \dfrac{p \sqsubseteq q,\ q \sqsubseteq r}{p \sqsubseteq r}$

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

(pre) $\quad \dfrac{p \sqsubseteq q}{\mu.p \sqsubseteq \mu.q}$

(rec) $\quad \dfrac{}{recP.p = p[recP.p/P]}$

$(inst)$ $\quad \dfrac{}{p\sigma \sqsubseteq q\sigma}$ for every inequation $p \sqsubseteq q$ and closed instantiation σ

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

$(cond1)$ $\quad \dfrac{[\![be]\!] = T}{be \longrightarrow p, q = p}$

$(cond2)$ $\quad \dfrac{[\![be]\!] = F}{be \longrightarrow p, q = q}$

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

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

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

Figure 4: The Proof System E_{rec}

We remind the reader that $V_n = \{v_1, \ldots, v_n\}$ is the set of the n first values. The compact approximations have the following fundamental property:

Theorem 4.2 *For all n and all $t \in Terms$*

1. *$t^{(n)} \in CoTerms(PN)$, i.e. $t^{(n)}$ is a syntactically compact term.*
2. *$\overline{ACT}[\![t]\!]\rho = \bigsqcup_n \overline{ACT}[\![t^{(n)}]\!]\rho$.*

Proof

1. May be proved by an induction on n combined with an inner structural induction.

2. In what remains of the proof we write $[\![_]\!]$ instead of $\overline{ACT}[\![_]\!]$. We have to prove that $\bigsqcup[\![t^{(n)}]\!]\rho \sqsubseteq [\![t]\!]\rho$ and $[\![t]\!]\rho \sqsubseteq \bigsqcup[\![t^{(n)}]\!]\rho$. To prove the first inequality it is sufficient to prove that $[\![t^{(n)}]\!]\rho \sqsubseteq [\![t]\!]\rho$ for all n. This may be proved in the same way as a similar property for the pure calculus given in Lemma 4.2.10 in [Hen88a]. We leave it to the reader to check the details. The proof for the opposite inequality, $[\![t]\!]\rho \sqsubseteq \bigsqcup[\![t^{(n)}]\!]\rho$, again follows the same pattern as the proof for a similar property given in Theorem 4.2.11 in [Hen88a]. The main difference is in connection with the restrictions on the values in the function and pair terms. As in the above mentioned reference we proceed by structural induction on t.

   $t = q \in Proc$: We proceed by case analysis on the form q takes. Here the only nontrivial case is $q = recP.p$. By definition $[\![recP.p]\!]\rho = YF$ where $F = \lambda a.[\![p]\!]\rho[P \mapsto a]$ and Y is the least fixed-point operator. Thus, to prove the result, it is sufficient to prove that $l = \bigsqcup_n [\![q^{(n)}]\!]\rho$ is a fixed point to F. So first let $\rho^n = \rho[P \mapsto [\![q^{(n)}]\!]\rho]$. Then $F([\![q^{(n)}]\!]\rho) = [\![p]\!]\rho^n$ and $[\![p^{(k+1)}]\!]\rho^k = [\![q^{(k+1)}]\!]\rho$ for all k. Now, as F is continuous, we have

$$F(\bigsqcup_n [\![q^{(n)}]\!]\rho) = \bigsqcup_n F([\![q^{(n)}]\!]\rho) = \bigsqcup_n [\![p]\!]\rho^n.$$

   By the structural induction

$$[\![p]\!]\rho^n = \bigsqcup_m [\![p^{(m)}]\!]\rho^n.$$

   We note that for any n, m and $k \geq Max\{m, n\}$,

$$[\![p^{(m)}]\!]\rho^n \sqsubseteq [\![p^{(k+1)}]\!]\rho^k.$$

   This implies

$$F(\bigsqcup[\![q^{(n)}]\!]\rho) = \bigsqcup_n(\bigsqcup_m [\![p^{(m)}]\!]\rho^n) =$$

$$\bigsqcup_k [\![p^{(k+1)}]\!]\rho^k = \bigsqcup_k [\![q^{(k+1)}]\!]\rho = \bigsqcup_n [\![q^{(n)}]\!]\rho$$

   which completes this case of the proof.

$t = [x]p \in Fun$: It is easy to see that

$$[\![[x]p^{(0)}]\!]\rho \sqsubseteq \cdots [\![[x]p^{(n)}]\!]\rho \sqsubseteq \cdots \sqsubseteq [\![[x]p]\!]\rho.$$

i.e. that $[\![[x]p]\!]\rho$ is an upper bound of the chain given above. We have to show that it is the least upper bound of the chain. So assume

$$[\![[x]p^{(0)}]\!]\rho \sqsubseteq \cdots [\![[x]p^{(n)}]\!]\rho \sqsubseteq \cdots \sqsubseteq f.$$

We have to show that $[\![[x]p]\!]\rho \sqsubseteq f$. So assume $v \in Val$. Then $v \in V_N$ for some N. Therefore for all $n \geq N$,

$$([\![([x]p)^{(n)}]\!]\rho)(v) = ([\![[x]x \in V_n \longrightarrow p^{(n)}, \Omega]\!]\rho)(v) =$$

$$[\![p^{(n)}[v/x]]\!]\rho \sqsubseteq f(v).$$

By the structural induction $[\![p[v/x]]\!]\rho$ is the least upper bound for the chain

$$[\![p^{(0)}[v/x]]\!]\rho \sqsubseteq [\![p^{(1)}[v/x]]\!]\rho \sqsubseteq \cdots \sqsubseteq [\![p^{(n)}[v/x]]\!]\rho \sqsubseteq \cdots.$$

This implies that $([\![[x]p]\!]\rho)(v) \sqsubseteq f(v)$. As $v \in Val$ was arbitrary this implies that $[\![[x]p]\!]\rho \sqsubseteq f$ as wanted.

$t = (v,q) \in Pairs$: May be proved in a similar way as the previous case and is left to the reader.

□

What remains of the section is devoted to the proof of the soundness and completeness of the proof system E_{rec} with respect to the model. To prove the completeness we introduce a notion of Ω-*normal forms* for compact terms and a corresponding normalization theorem.

Definition 4.3 [Ω-normal form] A compact term, $Cot \in CoTerms$, is said to be in a Ω-normal form if the following hold:

1. If $Cot = Cop \in CoProc$ then Cop has the form

$$\sum_i a_i.t_i\{+\Omega\}$$

where Ω is an optional summand and where t_i is in Ω-normal form. The empty sum is interpreted as NIL.

2. If $Cot = (e,Cop) \in CoPairs$ then $e = v \in Val$ and Cop is in a Ω-normal form.

3. If $Cot = [x]x : \underline{v_n} \longrightarrow \underline{p_n} \in Fun$ then p_i is in a Ω-normal form for $i \leq n$.

□

Lemma 4.4 *For all $Cot \in CoTerms$ there is Ω-normal form $n(Cot)$ such that $n(Cot) =_E Cot$*

Proof First we define the depth, $\delta(Cot)$ of a compact term Cot by

1. $\delta(NIL) = \delta(\Omega) = 0$
2. $\delta(Cop \setminus c) = \delta(Cop[R]) = \delta(Cop)$
3. $\delta(Cop_1 + Cop_2) = max\{\delta(Cop_i) | i \leq n\}$
4. $\delta(Cop_1 | Cop_2) = 1 + \delta(Cop_1) + \delta(Cop_2)$
5. $\delta(pre.Cot) = 1 + \delta(Cot)$
6. $\delta((e, Cop)) = \delta(Cop)$
7. $\delta([x].x : v_1, \cdots, v_n \longrightarrow Cop_1, \cdots, Cop_n) = max\{\delta(Cop_i) | i \leq n\}$

To prove the result we prove the following stronger result:

> For all $Cot \in CoTerms$ there is a Ω-normal form $n(Cot)$ such that $n(Cot) =_E Cot$ and $\delta(n(Cot)) \leq \delta(Cot)$.

We prove the statement by induction on $\delta(Cot)$. So assume that the statement holds for all Cot' with $\delta(Cot') < n$ and that $\delta(Cot) = n$. We will prove that the statement holds for Cot. We proceed by structural induction on Cot.

$Cot = Cop \in CoProc$: We proceed by a case analysis on the form Cop takes.

$Cop = NIL$: Trivial.

$Cop = \Omega$: We get the result by defining $n(\Omega) = NIL + \Omega$

$Cop = \mu.Cot, Cop_1 + Cop_2$: Follows from the structural induction and a simple use of the proof system.

$Cop = Cop_1 \setminus c$: By structural induction, Cop_1 has a Ω-normal form, onf, such that

$$Cop_1 =_E onf = \sum_i a_i.t_i\{+\Omega\} \text{ where } \delta(onf) \leq n$$

In particular $\delta(t_i) < n$ for all i. By a simple use of the proof system we get

$$Cop_1 \setminus c =_E \sum_{i:a_i \neq c} a_i.(t_i \setminus c)\{+\Omega\}$$

Furthermore $\delta(t_i \setminus c) = \delta(t_i) < n$. By the outer induction there are Ω-normal forms, $n_i =_E t_i \setminus c$, such that $\delta(n_i) \leq \delta(t_i \setminus c) < n$. By substitutivity

$$Cop_1 \setminus c =_E \sum_{i:a_i \neq c} a_i.n_i\{+\Omega\}$$

Obviously the right hand side of the equation is an Ω-normal form. Finally

$$\delta(\sum_{i:a_i \neq c} a_i.n_i\{\Omega\}) \leq 1 + \max_i\{\delta(n_i)\} \leq n$$

This completes the proof in this case.

$Cop = Cop_1[R]$: Similar.

$Cop = Cop_1|Cop_2$: This is the only non-trivial case. By induction Cop_1 and Cop_2 have Ω-normal forms, n_1 and n_2, with $\delta(n_i) \leq \delta(Cop_i), i = 1,2$. If either n_1 or n_2 is NIL, the result follows from Equation (NILpar) in Figure 3. Otherwise assume

$$n_1 = \sum_i a_i?.[x_i]r_i + \sum_j b_j!.(v_j, p_j) + \sum_k \tau.q_k\{+\Omega\}$$

and

$$n_2 = \sum_{i'} a'_{i'}?.[y_{i'}]r'_{i'} + \sum_{j'} b'_{j'}!.(v'_{j'}, p'_{j'}) + \sum_{k'} \tau.q'_{k'}\{+\Omega\}.$$

By substitutivity and the interleaving law

$$Cop = Cop_1|Cop_2 =_E n_1|n_2 =_E$$

$$INTL_{out}(n_1, n_2) + INTL_{in}(n_1, n_2) +$$

$$INTL_{\tau}(n_1, n_2) + COMM(n_1, n_2)\{+\Omega\}.$$

It is sufficient to prove that each of the summands can be reduced to an Ω-normal form with depth no greater than that of Cop. We only prove this for the summand $INTL_{in}$ as the proof of the statement for the remaining two is similar. We recall that

$$INTL_{in}(n_1, n_2) = INTL^l_{in}(n_1, n_2) + INTL^r_{in}(n_1, n_2)$$

where

$$INTL^l_{in}(n_1, n_2) = \textstyle\sum_i a_i?.[x_i](r_i|n_2)\{+\Omega\}$$

$$INTL^r_{in}(n_1, n_2) = \textstyle\sum_{i'} a'_{i'}?.[y_{i'}](n_1|r'_{i'})\{+\Omega\}.$$

We will only prove the statement for $INTL^l_{in}(n_1, n_2)$ as the proof for the other one is similar. Again it is sufficient to prove that the terms $[x_i](r_i|n_2)$ may be reduced to a Ω-normal form with depth strictly less than Cop. To prove this first we recall that r_i has the form $r_i = x_i : \underline{v_{n_i}}^i \longrightarrow \underline{p_{n_i}}^i$ where p^i_j is a Ω-normal form for $j \leq n_i$. Therefore we get

$$[x_i](r_i|n_2) = [x_i]((x_i : \underline{v_{n_i}}^i \longrightarrow \underline{p_{n_i}}^i)|n_2).$$

By using the rules (fun), (cond1) and (cond2) and substitutivity we obtain

$$[x_i](r_i|n_2) =_E [x_i]((x_i : \underline{v_{n_i}}^i \longrightarrow \underline{q_{n_i}}^i)$$

where $q_j^i = p_j^i|n_2$ for $j \leq n_i$. Furthermore

$$\delta(q_j^i) = \delta(p_j^i|n_2) \leq \max_{j \leq n_i}\{\delta(p_j^i|n_2)\} <$$

$$1 + \max_{j \leq n_i}\{\delta(p_j^i|n_2)\} =$$

$$1 + (1 + \max_{j \leq n_i}\{\delta(p_j^i)\} + \delta(n_2)) =$$

$$1 + (1 + \delta([x_i]r_i) + \delta(n_2)) =$$

$$1 + \delta(a_i?.[x_i]r_i) + \delta(n_2) \leq 1 + \delta(n_1) + \delta(n_2) \leq$$
$$1 + \delta(Cop_1) + \delta(Cop_2) = \delta(Cop) = n.$$

Thus the outer induction applies on q_j^i and we may conclude that $q_j^i =_E \gamma_j^i$ where γ_j^i is an Ω-normal form and such that $\delta(\gamma_j^i) \leq \delta(q_j^i)$. By substitutivity

$$INTL_{in}^l(n_1, n_2) =_E \sum_i a_i?.[x_i]x_i : \underline{v_n}^i \longrightarrow \underline{\gamma_n}^i$$

where the right hand side of the equality obviously is a Ω-normal form. Furthermore

$$\delta(\textstyle\sum_i a_i?.[x_i]x_i : \underline{v_n}^i \longrightarrow \underline{\gamma_n}^i) =$$

$$\max_i\{\delta(a_i?.[x_i]x_i : \underline{v_n}^i \longrightarrow \underline{\gamma_n}^i)\} =$$

$$1 + \max_i\{\max_j\{\delta(\gamma_j^i)\}\}) \leq$$

$$1 + \max_i\{\max_j\{\delta(q_j^i)\}\} \leq$$

$$\delta(Cop).$$

In a similar way we may show that $INTL_{in}^r(n_1, n_2)$ has a Ω-normal form with depth no greater than that of Cop and therefore that $INTL_{in}(n_1, n_2)$ has a Ω-normal form with depth no greater than $\delta(Cop)$. We can also prove a similar statement about $INTL_{out}(n_1, n_2)$, $INTL_\tau(n_1, n_2)$ and $COMM(n_1, n_2)$ and thereby for Cop.

$Cot \in CoPairs, CoFun$: Left to the reader. □

We will end this section by stating and proving the soundness and completeness of the proof system E_{rec} with respect to the denotational semantics.

Theorem 4.5 ***[Soundness and Completeness]*** *For all closed terms ct, cu in $CTerms$ we have*

$$ct \sqsubseteq_{E_{rec}} cu \text{ if and only if } ACT[\![ct]\!] \sqsubseteq ACT[\![cu]\!]$$

i.e. the proof system E_{rec} is sound and complete with respect to the denotational semantics.

Proof

Soundness: The soundness of the ω-rule is the content of Theorem 4.2 whereas the soundness of the (rec)-rule follows from the definition of the semantics of $rec.p$ as a least fixed point. What remains to prove is the soundness of E. We do this by reducing the proof to a proof of the soundness for syntactically compact terms with respect to $\overline{K}$. For this purpose we need the following property.

$$ct \sqsubseteq_E cu \Rightarrow \exists m \forall n \geq m.\, ct^{(n)} \sqsubseteq_E cu^{(n)} \tag{2}$$

This may be proved by induction on the depth of the proof for $ct \sqsubseteq_E cu$. The only non-trivial case is the base case when the interleaving law is used. We leave it to the reader to check the details of the proof.

The soundness of E over $CoTerms$ with respect to $\overline{K}$ follows easily from the definition of K and the fact that the elements of $CoTerms$ denote exactly $\overline{K}$. Now we may proceed as follows:

Assume $ct \sqsubseteq_E cu$. Then, by (2), $ct^{(n)} \sqsubseteq_E cu^{(n)}$ for all $n \geq m$ for some m. As $ct^{(n)}, cu^{(n)} \in CoTerms$, the soundness of E with respect to for these implies

$$\overline{K}[\![ct^{(n)}]\!] \prec_{\overline{K}} \overline{K}[\![cu^{(n)}]\!] \text{ for all } n \geq m$$

or equivalently

$$\overline{ACT}[\![ct^{(n)}]\!] \sqsubseteq \overline{ACT}[\![cu^{(n)}]\!] \text{ for all } n \geq m$$

Theorem 4.2 implies

$$ACT[\![ct]\!] \sqsubseteq ACT[\![cu]\!]$$

Completeness: Again we reduce the proof to proving that E is complete for $CoTerms$ with respect to $\overline{K}$. We first note that Theorem 4.2 and the ω-algebraicity of the model imply

$$\begin{aligned}
&\overline{ACT}[\![ct]\!] \sqsubseteq \overline{ACT}[\![cu]\!] && \Rightarrow \\
&\forall n.\overline{ACT}[\![ct^{(n)}]\!] \sqsubseteq \overline{ACT}[\![cu]\!] && \Rightarrow \\
&\forall n \exists m.\overline{ACT}[\![ct^{(n)}]\!] \sqsubseteq \overline{ACT}[\![cu^{(m)}]\!] && \Rightarrow \\
&\forall n \exists m.\overline{K}[\![ct^{(n)}]\!] \prec \overline{K}[\![cu^{(m)}]\!].
\end{aligned} \tag{3}$$

If E is complete for $CoTerms$ with respect to $\overline{K}$ then

$$\overline{K}[\![ct^{(n)}]\!] \prec \overline{K}[\![cu^{(m)}]\!] \Rightarrow ct^{(n)} \sqsubseteq_E cu^{(m)} \tag{4}$$

Now $cu^{(m)} \sqsubseteq_{E_{rec}} cu$ may easily be shown so (3), (4) and the ω-rule give

$$\overline{ACT}[\![ct]\!] \sqsubseteq \overline{ACT}[\![cu]\!] \Rightarrow \forall n.ct^{(n)} \sqsubseteq_{E_{rec}} cu \Rightarrow ct \sqsubseteq_{E_{rec}} cu.$$

Thus what remains to prove is the completeness of E on $CoTerms$ with respect to $\overline{K}$. By Lemma 4.4 and the soundness of E it is even enough to prove the completeness for Ω-normal forms with respect to $\overline{K}$ because:

Assume $\overline{K}[\![Cot_1]\!] \prec \overline{K}[\![Cot_2]\!]$. By Lemma 4.4 $Cot_i =_E n_i, i = 1,2$ where $n_i, i = 1,2$ are Ω-normal forms. By the soundness of E with respect to $\overline{K}$, $\overline{K}[\![n_i]\!] = \overline{K}[\![Cot_i]\!], i = 1,2$ and therefore $\overline{K}[\![n_1]\!] \prec \overline{K}[\![n_2]\!]$. If E is complete for Ω-normal forms with respect to $\overline{K}$ we may conclude that $n_1 \sqsubseteq_E n_2$. That $Cot_1 \sqsubseteq_E Cot_2$ follows from the transitivity of the proof system.

To prove the completeness for the Ω-normal forms we proceed as follows:

Assume n_1, n_2 are Ω-normal forms. We have to prove that

$$\overline{K}[\![n_1]\!] \prec \overline{K}[\![n_2]\!] \Rightarrow n_1 \sqsubseteq_E n_2$$

We proceed by structural induction on n_1.

$n_1 = NIL + \Omega$: Obvious.

$n_1 = NIL$: $\emptyset = \overline{K}[\![NIL]\!] \prec \overline{K}[\![n_2]\!]$ implies $\overline{K}[\![n_2]\!] = \emptyset$ and therefore that $n_2 = NIL$.

$n_1 = \sum_{i \leq n} \mu_i.t_i\{+\Omega\}$, $n \geq 1$: Then

$$\overline{K}[\![n_1]\!] = \{\langle \mu_i, \overline{K}[\![t_i]\!]\rangle | i \leq n\}[\cup\{\bot\}]$$

where $\bot \in \overline{K}[\![n_1]\!]$ if and only if Ω is a summand of n_1. As $\overline{K}[\![n_1]\!] \prec \overline{K}[\![n_2]\!]$ then $n_2 \neq NIL$ and $n_2 \neq \Omega$, i.e. n_2 has the form

$$n_2 = \sum_{j \leq m} \gamma_j.u_j\{+\Omega\}$$

and

$$\overline{K}[\![n_2]\!] = \{\langle \gamma_j, \overline{K}[\![u_j]\!]\rangle | j \leq m\}[\cup\{\bot\}]$$

where $m \geq 1$. Assume that $\langle \mu_i, \overline{K}[\![t_i]\!]\rangle \in \overline{K}[\![n_1]\!]$. Then $\mu_i = \gamma_{j_i}$ and $\overline{K}[\![t_i]\!] \prec \overline{K}[\![u_{j_i}]\!]$ for some $j_i \leq m$. By induction $t_i \sqsubseteq_E u_{j_i}$. As this holds for any i we get that

$$\sum_{i \leq n} \mu_i.t_i \sqsubseteq_E \sum_{i \leq n} \gamma_{j_i}.u_{j_i} \tag{5}$$

First assume that Ω is a summand in n_1. As obviously

$$\Omega \sqsubseteq_E \sum_j \gamma_j.u_j\{+\Omega\}$$

we get, by (5), substitutivity and absorption of the proof system, that

$$n_1 = \sum_i \mu_i.t_i + \Omega \sqsubseteq_E \sum_i \gamma_{j_i}.u_{j_i} + \sum_j \gamma_j.u_j\{+\Omega\} =_E n_2$$

which proves the statement in this case. Next assume that Ω is not a summand in n_1. This implies that $\bot \notin \overline{K}[\![n_1]\!]$ which in turn implies that $\bot \notin \overline{K}[\![n_2]\!]$. We may therefore

conclude that Ω is not a summand of n_2 either. In a similar way as before we get

$$\sum_{j \leq m} \mu_{i_j}.t_{i_j} \sqsubseteq_E \sum_{j \leq m} \gamma_j.u_j \tag{6}$$

where $\{i_j | j \leq m\} \subseteq \{1, \ldots n\}$. Now from (5), (6), the absorption and the substitutivity of the proof system we get

$$n_1 = \textstyle\sum_{i \leq n} \mu_i.t_i =_E \sum_{i \leq n} \mu_i.t_i + \sum_{j \leq m} \mu_{i_j}.t_{i_i}$$

$$\sqsubseteq_E \textstyle\sum_{i \leq n} \gamma_{j_i}.u_{j_i} + \sum_{j \leq m} \gamma_j.u_j = \sum_{j \leq m} \gamma_j.u_j = n_2$$

which completes the proof for this case.

$n_1 \in CoFun, CoPair$: Follows easily from the induction.

□

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 Σ-orders have been extended to the so-called applicative signatures and (Σ, C)-algebras and (Σ, C)-orders. Furthermore we show how we may take 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. The language is obtained as an instantiation of the general class of languages we defined where the signature Σ is taken to be the set of the standard operators of CCS.

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. It is a direct extension of the model defined by Abramsky in [Abr91] and a modification of the model given by Milne and Milner in [MM79]. As all the constructions we use in the definition of the equation are standard and well known to preserve *cpo*s and ω-algebraicity the solution we obtain is an ω-algebraic *cpo*.

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

We have also presented an equationally based proof system and shown its soundness and completeness with respect to the model. The algebraicity of the model and the way we define the operators makes it possible for us to reduce the proof of the soundness and completeness to a proof of the same property on a sub-language of the actual language, the so-called compact terms. This is an inductively defined language which denotes exactly the compact elements of 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 companion paper [Ing95a] 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 determined by the compact elements. Behaviourally this can be interpreted as meaning that the preorder may be obtained by some kind of finite observations. This is not the case for bisimulation as it is well known even for the pure calculus. In [Ing95a] 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 it.

References

[Abr87] S. Abramsky. Domain theory in logical form. In *Proceedings* 2^{nd} *Annual Symposium on Logic in Computer Science, Ithaca, New York*. IEEE Computer Society Press, 1987. Full version to appear in *Annals of Pure and Applied Logic*.

[Abr91] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.

[GTWW77] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *Journal of the ACM*, 24(1):68–95, 1977.

[Hen88a] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, Cambridge, Massachusetts, 1988.

[Hen94] M. Hennessy. A fully abstract denotational model for higher-order processes. *Information and Computation*, 112(1):55–95, July 1994.

[HI93] M. Hennessy and A. Ingólfsdóttir. A theory of communicating processes with value-passing. *Information and Computation*, 107(2):202–236, 1993.

[HL93a] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. In E. Best, editor, *Proceedings CONCUR 93, Hildesheim*, volume 715 of *Lecture Notes in Computer Science*, pages 202–216. Springer-Verlag, 1993.

[HL93b] M. Hennessy and X. Liu. A modal logic for message passing processes. In C. Courcoubetis, editor, *Proceedings of the Fifth International Conference Computer Aided Verification, Elounda, Greece, July 1993*, volume 697 of *Lecture Notes in Computer Science*, pages 359–370. Springer-Verlag, 1993.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[HP80] M. Hennessy and G.D. Plotkin. A term model for CCS. In P. Dembiński, editor, 9^{th} *Symposium on Mathematical Foundations of Computer Science*, volume 88 of *Lecture Notes in Computer Science*, pages 261–274. Springer-Verlag, 1980.

[Ing93] A. Ingólfsdóttir. Late and early semantics coincide for testing. Report 93–2008, Institute for Electronic Systems, Department of Mathematics and Computer Science, Aalborg University Centre, 1993. To appear in *Theoretical Computer Science*.

[Ing94] A. Ingólfsdóttir. *Semantic Models for Communicating Process with Value-Passing*. PhD thesis, School of Cognitive and Computing Sciences, University of Sussex, June 1994. Computer Science Report 8/94. Also available as Report R–94–2044, Department of Mathematics and Computer Science, Aalborg University.

[Ing95a] A. Ingólfsdóttir. A Semantic Theory for Value-Passing Processes, Late Approach, Part II: A Behavioural Semantics and Full Abstractness To appear as Technical Report, BRICS, Aalborg University.

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

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

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Department of Computer Science, University of Edinburgh, October 1991. To appear in *Proceedings of the International Summer School on Logic and Algebra of Specification*, Marktoberdorf, August 1991.

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

[MPW91] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings CONCUR 91, Amsterdam*, volume 527 of *Lecture Notes in Computer Science*, pages 45–60. Springer-Verlag, 1991.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.

[Plo76] G.D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5:452–487, 1976.

[Plo81] G.D. Plotkin. Lecture notes in domain theory, 1981. University of Edinburgh.

[San93] D. Sangiorgi. From π-calculus to higher-order π-calculus — and back. In *TAPSOFT '93*, volume 668 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[Smy78] M. Smyth. Powerdomains. *Journal of Computer and System Sciences*, 16(1), 1978.

[Tho89] B. Thomsen. Plain CHOCS. Report DOC 89/4, Department of Computing, Imperial College, 1989.

[Win85] G. Winskel. On powerdomains and modality. *Theoretical Computer Science*, 36:127–137, 1985.

Recent Publications in the BRICS Report Series

RS-95-3 Anna Ingólfsdóttir. *A Semantic Theory for Value–Passing Processes, Late Approach, Part I: A Denotational Model and Its Complete Axiomatization*. January 1995. 37 pp.

RS-95-2 François Laroussinie, Kim G. Larsen, and Carsten Weise. *From Timed Automata to Logic - and Back*. January 1995. 21 pp.

RS-95-1 Gudmund Skovbjerg Frandsen, Thore Husfeldt, Peter Bro Miltersen, Theis Rauhe, and Søren Skyum. *Dynamic Algorithms for the Dyck Languages*. January 1995. 21 pp.

RS-94-48 Jens Chr. Godskesen and Kim G. Larsen. *Synthesizing Distinguishing Formulae for Real Time Systems*. December 1994. 21 pp.

RS-94-47 Kim G. Larsen, Bernhard Steffen, and Carsten Weise. *A Constraint Oriented Proof Methodology based on Modal Transition Systems*. December 1994. 13 pp.

RS-94-46 Amos Beimel, Anna Gál, and Mike Paterson. *Lower Bounds for Monotone Span Programs*. December 1994. 14 pp.

RS-94-45 Jørgen H. Andersen, Kåre J. Kristoffersen, Kim G. Larsen, and Jesper Niedermann. *Automatic Synthesis of Real Time Systems*. December 1994. 17 pp.

RS-94-44 Sten Agerholm. *A HOL Basis for Reasoning about Functional Programs*. December 1994. PhD thesis. viii+224 pp.

RS-94-43 Luca Aceto and Alan Jeffrey. *A Complete Axiomatization of Timed Bisimulation for a Class of Timed Regular Behaviours (Revised Version)*. December 1994. 18 pp. To appear in *Theoretical Computer Science*.

RS-94-42 Dany Breslauer and Leszek Gąsieniec. *Efficient String Matching on Coded Texts*. December 1994. 20 pp.