with each other using a form of instantaneous exchange of messages called *handshake communication*. Each individual process in a program may be viewed as a straightforward imperative program working on its own memory, with assignment statements, boolean tests, iteration, etc.

Subsequently, although some work was done on the semantics of this language, [FLP 84], most of the research effort was devoted to semantic theories of more abstract versions, such as *theoretical CSP, TCSP*, [BHR 84], [Plo 82] and *CCS*, [Mil 89]. These languages, often refered to as *process algebras*, differ significantly from the original *CSP*. For example *TCSP* may be viewed as an applicative language. There is no store nor assignment statement and no values may be transmitted between processes; they only communicate by synchronising on signals. Nevertheless this area of research has been very fruitful. Semantic the

on
CSP

The language is described in detail in Section 2.1 where it is compared with *Occam* and, in particular, the version of *Occam* used in [HR 88] and [Ros 87].

## 1.2. Denotational Models

A standard model for *TCSP* is the *failure-sets model* described, for example in [BHR 84]. It consists of suitable collections of pairs of the form $(s, X)$ where $s$ is a trace of actions which the process can perform and $X$ is a finite set of actions which it can subsequently refuse. A further component of the model also contains information about possible internal divergences.

This model has been adapted in [Ros 87] in order to interpret a subset of *Occam*. The adaptation is two-fold: the first to handle the communication of values and the second to handle stores. Values are accommodated by using actions of the form $c.v$ where $c$ is a channel name and $v$ is a value. Intuitively this action represents the passage of the value $v$ along the channel $c$. Stores are incorporated by extending that part of the model which records possible internal divergences. Whereas previously it consisted simply of the traces of a process which could lead to divergence, it also now includes those traces which lead to successful termination and the resulting store.

There are two major problems associated with this extended model, both connected with the treatment of value-passing. The first is that the allowed set of values must be finite for otherwise the semantic operators would no longer be continuous. Although in practice a given program will only ever use a finite set of values, it is conceptually very restricting to limit the possible values usable in a programming language to be finite. The second arises from the form of the actions, $c.v$. Processes *input* values from channels or *output* values to channels and these are different actions. In [Ros 87] they are modelled by the same action, $c.v$ which merely records the passage of $v$ along $c$. This is possible because the

private store and the only access to this store is by communication with the process. From this point of view the two processes

$$(x := 1).c?x.P$$

and

$$(x := 2).c?x.P$$

are behaviourally identical; although they have different effects on their private memory, this difference is not discernible to any external observer or, indeed, any larger system which uses them as subprocesses. Of course it would be quite different if, after updating its private memory, there is a subsequent possibility of communicating the effect of this update. So, for example, the processes

$$(x := 1).c!x.STOP$$

and

$$(x := 2).c!x.STOP$$

will be distinguished in the model. But in general there is no need to record the sequence of store transformations carried out by a process as it receives and

syntactic constructs we borrow from process algebras were specifically designed

which is supposed to be a choice between two behaviours, one which outputs the value of $x$, which is 0, and the other which updates the store. Our solution to this problem is given in Section 3.1 where we present the operational semantics. This is followed in Section 3.2 by a definition of testing and the associated preorder. Finally, in Section 3.3, we prove that our model is fully-abstract.

## 2. The Language and its Model

In this section we introduce our language and describe its denotational semantics. The section is divided into five subsections: In the first one we give the syntax of the language and some example programs. In §2.2 we describe a general mathematical model for the language in terms of natural interpretations and then in §2.3 we give a brief introduction to one such model, the so-called *Strong Acceptance Trees* [HI 89], which is a modification of the model Strong Acceptance Trees introduced in [He 85] and [He 88]. In §2.4 we define syntactically and semantically finite approximations of programs and show that the denotational interpretation is completely decided by these. In the last subsection we define a proof system and prove its soundness and completeness with respect to the model.

### 2.1. Syntax

Our language, $VPLA$, (a Value-Passing Language with Assignment) is an extension of the applicative concurrent language, $VPL$, introduced in [HI 89]. We have added the imperative construct assignment but only as a form of prefixing. As we want to compare our language with the existing concurrent programming language *Occam* we omit renaming.

The language is therefore a slight modification of *Occam* although we use a different more abstract syntax. The main differences are that beside the usual external choice or alternation operator $ALT$, which in our setting is called $+$, we have an internal choice operator $\oplus$. Further we have the restriction in our language that sequential composition is not allowed in general but only as a prefixing of the input/output actions or of an assignment statement. The operator $\backslash c$ plays the role of a local declarations of channelson6ec.9.3Td

$$t \quad ::= \quad op(t_1,\ldots,t_k), op \in \Sigma_k \;\mid\; P \mid pre.t \mid recP.t \mid be \rightarrow t,t$$

$$pre \quad ::= \quad c!e \mid c?x \mid x := e$$

**Fig. 1.** Syntax

and a boolean expression, $FVar(be)$, to be predefined. We let $Chan$ denote a predefined set of *channel names*, ranged over by $c$.

The allowed operators in the syntax are $STOP$, and $\Omega$ of arity 0, $\backslash c$ of arity 1 and $\oplus$, $+$ and $|$ of arity 2. We use $\Sigma$ to denote this collection of operators and $\Sigma_k$ those of arity $k$. We also need a predefined set of process names, $PN$, ranged over by upper-case letters such as $P$, $Q$, etc. The set of terms is then defined by the BNF-definition given in Figure 1.

iii) $in_D : (Chan \times (Val \longrightarrow D)) \longrightarrow D$ is a total function continuous in its second argument, where $Val \longrightarrow D$ inherits the natural pointwise ordering from $D$.

Given such a natural interpretation, $D$, we can define a semantic interpretation of *VPLA* following the usual approach of denotational semantics. We let $Env_D$ be the set of $D$-environments, i.e. mappings from $PN$ to $D$, ranged over by $\rho$ and $St$ the set of stores, mappings from $Var$ into $Val$, ranged over by $\sigma$. We assume evaluation functions $[\![\ \ ]\!] : Exp \longrightarrow (St \longrightarrow Val)$ and $[\![\ \ ]\!] : BExp \longrightarrow (St \longrightarrow \{T, F\})$. Then the semantics of the language *VPLA* is given as a function:

$$D[\![\ ]\!] : VPLA \longrightarrow (Env_D \longrightarrow (St \longrightarrow D))$$

and is defined by structural induction on *VPLA*:

i) $D[\![P]\!]\rho\sigma = \rho(P)$

ii) $D[\![op(\underline{t})]\!]\rho\sigma = op_D(D[\![\underline{t}]\!]\rho\sigma)$

iii) $D[\![recP.t]\!]\rho\sigma = Y\lambda d.D[\![t]\!]\rho[d/P]\sigma$

iv) $D[\![be \rightarrow t, u]\!]\rho\sigma = \begin{array}{ll} D[\![t]\!]\rho\sigma & \text{if } [\![be]\!]\sigma = T \\ D[\![u]\!]\rho\sigma & \text{if } [\![be]\!]\sigma = F \end{array}$

v) $D[\![c!e.t]\!]\rho\sigma = out_D(c, [\![e]\!]\sigma, D[\![t]\!]\rho\sigma)$

vi) $D[\![c?x.t]\!]\rho\sigma = in_D(c, \lambda v.D[\![t]\!]\rho\sigma[v/x])$

vii) $D[\![x := e.t]\!]\rho\sigma = D[\![t]\!]\rho\sigma[[\![e]\!]\sigma/x]$

where $Y$ is the least-fixpoint operator for continuous functions over $D$.

## 2.3. Acceptance Trees

In this subsection we will give a brief description of the mathematical model called *Strong Acceptance Trees*, or just *Acceptance Trees*, introduced in [He 85] and explained in more detail in [He 88]. Then we will explain the modified version, which models the value-passing calculus $VPL$ in [HI 89]. The definition of the pure version assumes a set of "pure" actions, $Act$ which processes can perform. Further we need the notion of saturated sets. Thus a finite set, $\mathcal{A} \subseteq \mathcal{P}_{fin}(Act)$, where $\mathcal{P}_{fin}(Act)$ is the family of finite subsets of $Act$, is said to be saturated if it satisfies the following conditions:

1. $\bigcup \mathcal{A} \in \mathcal{A}$
2. $A, B \in \mathcal{A}$ and $A \subseteq C \subseteq B$ implies $C \in \mathcal{A}$.

The set of all saturated subsets of $\mathcal{P}_{fin}(Act)$, all saturated sets over $Act$, is denoted by $sat(Act)$. A saturated set over $Act$ is called an acceptance set. The saturated closure, $c(\mathcal{A})$, of a finite set, $\mathcal{A} \subseteq \mathcal{P}_{fin}(Act)$ is defined as the least set which satisfies

1. $\mathcal{A} \subseteq c(\mathcal{A})$
2. $\bigcup \mathcal{A} \in c(\mathcal{A})$

3. $A, B \in c(\mathcal{A})$ and $A \subseteq C \subseteq B$ implies $C \in c(\mathcal{A})$.

It follows easily from the definition, that $c(\mathcal{A})$ is the least saturated set, which includes $\mathcal{A}$.

Finite acceptance trees are rooted finite branching trees of finite depth. Each node is either open or closed. Each closed node is labelled by an acceptance set, $\mathcal{A}$, where each set, $A$, in $\mathcal{A}$ models a state the process can reach internally or without performing any visible action. The actions in $A$ are those which the process can perform when in that state. The arcs leading from a node are labelled by the actions which occur in the acceptance set labelling the node.

1. $D = (H($

We also need a definition of $AT^v_n$, the *Acceptance Trees of depth and functional size n*. We let $Fin_n(Val \longrightarrow D) = \{f \in Fin(Val \longrightarrow D) | f(v) = \bot \text{ whenever } v \notin V_n\}$ and $G^n_{fin} = Fin_n(Val \longrightarrow D) \uplus Fin(Val \rightharpoonup D)$.

**Definition 2.3.2.** We define $AT^v_{(n)}$ by induction on $n$ by

1. $\{\bot\} \in AT^v_{(n)}$ for all $n$
2. if $\mathcal{A} \in sat(Ev)$ and $f : \bigcup \mathcal{A} \longrightarrow G^n_{fin}(Val, AT_{(n)})$ then $(\mathcal{A}, f) \in AT^v_{(n+1)}$.

We have the following result:

**Lemma 2.3.1.**

1. $AT^v_0 \subseteq AT^v_1 \subseteq \cdots \subseteq$ 100211()201220120[()()02 100211()2002=10001

By induction $(f(e))(v) \leq (f_{m(e,v)}(e))(v)$ for some $m(e,v)$ for all $e \in \bigcup \mathcal{A}$ and $v \in domain(f(e))$. As $domain(f(e))$ and $\bigcup \mathcal{A}$ are finite, the set $E = \{(e,v) \mid e \in \bigcup \mathcal{A}, v \in domain(f(e))\}$ is finite. We can therefore define

$$N = max\{m(e,v) \mid (e,v) \in E\}$$

and we get easily that $f \leq f_N$ and the result follows.

Next we will prove that $fAT^v$ are the only finite elements in $AT^v$. For this purpose for any $T \in AT^v$ we define $T^{(n)}$, the projection of $T$ on $AT^v_{(n)}$. This we do in the following way:

1. $T^{(0)} = \bot$

2.a $\bot^{(n+1)} = \bot$

2.b $(\mathcal{A}, f)^{(n+1)} = (\mathcal{A}, f^{(n)})$, $n = 1, 2, \ldots$

where $f^{(n)}(c?)$ is defined by

$$(f^{(n)}(c?))(v) = \begin{cases} (f(c?)(v))^{(n)} & \text{if } v \in V_n \\ \bot & \text{otherwise} \end{cases}$$

and $f^{(n)}(c!)$ is defined by

$$domain(f^{(n)}(c!)) = domain(f(c!)) \cap V_n$$
$$(f^{(n)}(c!))(v) = ((f(c!))(v))^{(n)} \text{ for } v \in domain(f^{(n)}(c!))$$

By an easy induction we get that $T^{(n)} \in AT^v_{(n)}$ and that $T^{(0)} \leq T^{(1)} \leq \cdots T^{(n)} \leq T$ for all $n$. We will prove that $T$ is the *lub* for the chain. To do so we have to refer to some general results about initial solutions in $\underline{CPO}$ ([Plo 81]).

Now we recall the standard definition of $T^n$, the $n$-th approximation of $T$:

1. $T^0 = \bot$

2.a $\bot^{n+1} = \bot$

2.b $(\mathcal{A}, f)^{n+1} = (\mathcal{A}, f^n)$, $n = 1, 2 \ldots$

where $f^n(c?)$ is defined by

$$f^n(c?) = (f(c?)(v))^n$$

and $f^n(c!)$ is defined by

$$domain(f^n(c!)) = domain(f(c!))$$
$$(f^n(c!))(v) = ((f(c!))(v))^n \text{ for } v \in domain(f^n(c!))$$

Note that the $n$th approximation, $T^n$, is in general not a finite element of the model. From the general theory we know that $T = \bigsqcup_n T^n$. Furthermore we can show that $\bigsqcup_n T^{(n)} = \bigsqcup_n T^n$. The "$\leq$" part follows from the obvious fact that $T^{(n)} \leq T^n$ for all $n$. We get the "$\geq$" part by showing that $T^m \leq \bigsqcup_n T^{(n)}$ for all $m$ by induction on $m$ as follows:

1. $T^0 \leq \bigsqcup_n T^{(n)}$ is obvious
2. Now we want to prove for any $T$ that $T^{m+1} \leq \bigsqcup_n T^{(n)}$ given the statement is true for $m$. If $T = \bot$ we are done so assume $T$ has the form $(\mathcal{A}, f)$. Then $T^{m+1} = (\mathcal{A}, f^m)$ and $T^{(n)} = (\mathcal{A}, f^{(n-1)})$ for $n = 1, \ldots$. We also can easily prove that $\bigsqcup_n (\mathcal{A}, f^{(n)}) = (\mathcal{A}, \bigsqcup_n f^{(n)})$ where $\bigsqcup_n f^{(n)}$ is defined by

terms" to distinguish them from the usual syntactically finite terms. We will in the following define formally the semantically finite terms and then show that they denote exactly the finite elements in the model, i.e. $fAT^v$.

**Definition 2.4.1.** We define the set of semantically finite terms as the least set, $SF$, which satisfies:

1. $STOP, \Omega \in SF$

case $c?x.t$ as in the first case we have $(c?x.t)^n = c?x.t^n$. We will now show that the interpretation of a term in $AT^v$ is completely defined by the meaning of the semantically finite approximations of the term. This is the content of the following theorem:

**Theorem 2.4.1.** For all $t \in VPLA, \sigma \in St$ and $\rho \in Env$

$$[\![t]\!]\rho\sigma = \bigsqcup_n [\![t^{(n)}]\!]\rho\sigma$$

*Proof.* The proof is very similar to the corresponding one for Theorem 4.2.11 in [He 88]. As there we proceed by structural induction. The only case which is different is $t = c?x.u$ and will therefore be given in detail.

By definition of $[\![\_ \quad )$

$$
\begin{array}{rcl}
X \oplus (Y \oplus Z) & = & (X \oplus Y) \oplus Z \\
X \oplus Y & = & Y \oplus X \\
X \oplus X & = & X \\
X + (Y + Z) & = & (X + Y) + Z \\
X + Y & = & Y + X \\
X + X & = & X \\
X + STOP & = & X \\
pre.X + pre.Y & = & pre.(X \oplus Y) \\
c?x.X + c?x.Y & = & c?x.X \oplus c?x.Y \\
c!e.X + c!e'.Y & = & c!e.X \oplus c!e'.Y \\
X + (Y \oplus Z) & = & (X + Y) \oplus (X + Z) \\
X \oplus (Y + Z) & = & (X \oplus Y) + (X \oplus Z) \\
X \oplus Y & \sqsubseteq & X \\
X + \Omega & \sqsubseteq & \Omega \\
\Omega & \sqsubseteq & X \\
(X \oplus Y) \setminus c & = & X \setminus c \oplus Y \setminus c \\
(X + Y) \setminus c & = & X \setminus c + Y \setminus c \\
(pre.X) \setminus c & = & \begin{cases} pre.(X \setminus c) & \text{if } c \neq chan(pre) \\ STOP & \text{otherwise} \end{cases} \\
STOP \setminus c & = & STOP \\
\Omega \setminus c & = & \Omega \\
(X \oplus Y) \mid Z & = & X \mid Z \oplus Y \mid Z \\
X \mid (Y \oplus Z) & = & X \mid Y \oplus X \mid Z \\
STOP \mid X & = & X \mid STOP \; = \; X \\
X \mid (Y + \Omega) & = & (X + \Omega) \mid Y \; = \; \Omega
\end{array}
$$

**Fig. 2.** Equations

processes, the interpretation in the model is independent of the store and the environment and thus we can may write $[\![p]\!]$ instead of $[\![p]\!]\sigma\rho$.

The proof system is equationally based and is given in Figure 5 and the equations are given in Figure 2, 3 and 4. In the interleaving law in Figure 4 the predicate $comms(X, Y)$ is defined to be true if $X$ and $Y$ can communicate and false otherwise. So it is true only if there is a pair $a_i, b_j$ of complementary actions, one of the form $c?x$ and the other $c!e$. In Rule V of Figure 5 we use $\eta$ to

$$
\begin{array}{rcl}
(x := e).STOP & = & STOP \\
(x := e).\Omega & = & \Omega \\
(x := e).X \oplus Y & = & (x := e).X \oplus (x := e).Y \\
(x := e).X + Y & = & (x := e).X + (x := e).Y \\
(x := e).X \mid Y & = & (x := e).X \mid (x := e).Y \\
(x := e).c!e'.X & = & c!e'[e/x].(x := e).X \\
(x := e).c?y.X & = & c?y.(x := e).X,\ x \neq y, y \text{ not free in } e \\
(x := e).be \rightarrow X, Y & = & be[e/x] \rightarrow (x := e).X,\ (x := e).Y
\end{array}
$$

**Fig. 3.** Equations for Assignment

range over arbitrary substitutions of terms for variables. As usual we define $\sqsubseteq_E$ as the least relation which satisfies the rules and equations in Figures 2-5, and $t =_E u$ means $t \sqsubseteq_E u$ and $u \sqsubseteq_E t$.

The system is basically the same as the one for the applicative language $VPL$ in [HI 89]. We only have added equations, Figure 3, to reason about assignment and one new inference rule, the second part of rule VII in Figure 5, to assure substitutivity for assignment. In Figure 3 the assignment prefixing does not affect the meaning of the processes $STOP$ and $\Omega$. This reflects our ideas that stores and therefore bindings of variables to values can only be investigated by communication. Thus changing the value binding of a variable does not affect the process if it is not able to output the value of that variable. Further assignment distributes over the operators + and | which reflects our ideas of each subcomponent of the system having their private store only accessible by others via communication.

The new rules allow us to remove the assignments from any finite term and prove it equal to an assignment free finite term, i.e. a term in $VPL$.

**Lemma 2.5.1.** For all finite $d \in VPLA$ there is a finite $d' \in VPL$ such that $d = d'$.

*Proof.* Follows easily by structural induction on $d$ and a repeated use of the equations in Figure 3 and Rule IX. □

Another significant difference from the proof system in [HI 89] is the definition of the finite approximations $t^{(n)}$ in the $\omega$-rule (rule VI) as they are now only allowed to use a finite number of values. In [HI 89] we defined the $n$-th approximation of the term $c?x.u$ by $(c?x.u)^n = c?x.u^n$. Then we proved the completeness of the proof system by means of normal forms and head normal forms. We could have used the same procedure for the completeness proof in

Let $X, Y$ denote $\sum\{a_i.X_i, i \in I\}, \sum\{b_j.Y_j, j \in J\}$ where the same data variables do not occur in $X$ and $Y$. Then

$$X \mid Y = \begin{cases} ext(X,Y) & \text{if } comms(X,Y) = false \\ (ext(X,Y) + int(X,Y)) \oplus int(X,Y) & \text{otherwise} \end{cases}$$

where

$$ext(X,Y) = \sum\{a_i.(X_i \mid Y),\ i \in I\} + \sum\{b_j.(X \mid Y_j),\ j \in J\}$$
$$int(X,Y) = \sum\ \{X_i[v/x] \mid Y_j,\ a_i = c?x,\ b_j = c!v\}$$
$$\oplus \sum\ \{X_i \mid Y_j, [v/y]\ a_i = c!v,\ b_j = c?y\}$$

**Fig. 4.** Interleaving Law

| | | |
|---|---|---|
| I | $t \sqsubseteq t$ | $\dfrac{t \sqsubseteq u,\ u \sqsubseteq v}{t \sqsubseteq v}$ |
| II | $\dfrac{t_i \sqsubseteq u_i}{op(\underline{t}) \sqsubseteq op(\underline{u})}$ for every $op \in \sum$ | |
| III | $\dfrac{t \sqsubseteq u}{c!e.t \sqsubseteq c!e.u}$ | $\dfrac{t[v/x] \sqsubseteq u[v/x] \text{ for every } v \in V}{c?x.t \sqsubseteq c?x.u}$ |
| IV | $\dfrac{t \sqsubseteq u}{recP.t \sqsubseteq recP.u}$ | $\dfrac{}{recP.t = t[recP.t/P]}$ |
| V | $\dfrac{t \sqsubseteq u}{t\eta \sqsubseteq u\eta}$ | $\dfrac{}{t \sqsubseteq u}$ for every equation $t \sqsubseteq u$ |
| VI | $\dfrac{\forall n.t^{(n)} \sqsubseteq u}{t \sqsubseteq u}$ | |
| VII | $\dfrac{[\![e]\!] = [\![e']\!]}{c!e.t = c!e'.t}$ | $\dfrac{[\![e]\!] = [\![e']\!]}{x := e.t = x := e'.t}$ |
| VIII | $\dfrac{[\![be]\!] = T}{be \rightarrow t, u = t}$ | $\dfrac{[\![be]\!] = F}{be \rightarrow t, u = u}$ |
| IX | $\dfrac{}{c?x.t = c?y.t[y/x]}$ if $y$ does not occur free in $t$ | |

**Fig. 5.** Proof System

this case; instead we have chosen to define the $n$-th approximations $t^{(n)}$ in this slightly different way which can be seen in Definition 2.4.2. Unlike in the previous paper $t^{(n)}$ is semantically finite and thus the new $\omega$-rule

$$\frac{\forall n.\, t^{(n)} \sqsubseteq u}{t \sqsubseteq u}$$

reflects the $\omega$-algebraicness in the model. We can therefore use the completeness of the old system for finite terms and then take advantage of the $\omega$-algebraicness of the model to prove the completeness in the general case.

Obviously the original definition of finite approximations, $t^n$ in [HI 89], dominates the new ones, i.e. we can prove that for all $t \in VPLA$, $t^{(n)} \sqsubseteq_E t^n$ without using the $\omega$–rule and therefore the new $\omega$-rule implies the original one

$$\frac{\forall n.\, t^n \sqsubseteq u}{t \sqsubseteq u}$$

The new system is thus stronger than the old one, and a completeness for the old one implies completeness for the new one. We will take advantage of this fact to prove the completeness of the new proof system.

We end this section by stating and proving the soundness and completeness of the system for processes with respect to $AT^v$. We start with the soundness.

**Theorem 2.5.1. (Soundness)** For all $t, u$

$$t \sqsubseteq_E u \text{ implies } [\![t]\!] \leq [\![u]\!].$$

*Proof.* The soundness is already proved for most of the rules and equations in [HI 89]. The soundness of the $\omega$-rule is the content of Theorem 2.4.1 and the soundness of the remaining ones is obvious. □

For the completeness we start by proving the result for finite processes and then show how the general result follows from this as an easy corollary.

**Lemma 2.5.2.** For all finite $d_1$ $VuProof\beta_{s}uTjR\psi\psi Tf\psi\psi TdETj\psi Tf\psi\psi Td\psi TduTjTJR\psi\psi Tf\psi\psi TdTjR\psi\psi Tf\psi$

Lemma and the $\omega$-algebraicity of the model. Thus assume that $[\![p]\!] \leq [\![q]\!]$. From Lemma 2.4.1 we get that $[\![p]\!] = \bigsqcup_n [\![p^{(n)}]\!]$ and $[\![q]\!] = \bigsqcup_n [\![q^{(n)}]\!]$. Therefore $\bigsqcup_n [\![p^{(n)}]\!] \leq \bigsqcup_n [\![q^{(n)}]\!]$ which implies that

$$\forall k.\, [\![p^{(k)}]\!] \leq \bigsqcup_n [\![q^{(n)}]\!]$$

As $[\![p^{(k)}]\!]$ is a finite element of the model we have

$$\forall k \exists m.\, [\![p^{(k)}]\!] \leq [\![q^{(m)}]\!].$$

By the already proved completeness for finite terms

$$\forall k \exists m.\, p^{(k)} \sqsubseteq_E q^{(m)}.$$

As we know that $q^{(m)} \sqsubseteq_E q$ this implies

$$\forall k .p^{(k)} \sqsubseteq_E q$$

and the result follows from the $\omega$-rule. □

## 3. Operational Semantics and Full Abstractness

In this section we define an operational semantics for our language and introduce the notion of testing. The operational semantics is defined in such a way that it captures our intuition of the behaviour of processes or configurations described in the introduction, i.e. that each subprocess of a system has its own private store only accessible by other processes by means of communication. For a further

In following this standard approach a problem arises when we have to deal with these operators. In both cases each of the components can update the store internally and thus affect the variable bindings for the other. Let us have a look at an example. We use the notation $\sigma[x_1/v_1, \ldots, x_n/v_n]$ for the store $\sigma'$ defined by $\sigma'(x) = v_i$ if $x = x_i, i = 1, \ldots, n$ and $\sigma'(x) = \sigma(x)$ otherwise. Let

choice outlined in the introduction. Furthermore this solution coincides with the definition of the denotational semantics in the sense that there the stores are distributed over the operators and assignment in one component does not have any influence on the semantics of another component.

In the following we will try to formalise these informal ideas of the operational behaviour of processes, or more precisely configurations.

As motivated above, we need to introduce external choice and parallel composition between configurations. Thus more complex configurations are built up from the basic ones, which only consist of a pair of a term and a store. To simplify the rules for the operational semantics we also introduce the operators $\oplus$, $\backslash c$ and prefixing for configurations. This leads to the following definition of configurations:

**Definition 3.1.1.** The set of basic configurations, BCon, is defined as:

$$BCon = \{\langle t, \sigma\rangle \mid t \in VPLA, \sigma \in St\}.$$

Now we define the set of configurations, $Con$, as the least set, which satisfies

1) $BCon \subseteq Con$
2) $\langle t, \sigma\rangle \in BCon$ implies $pre.\langle t, \sigma\rangle \in Con$
3) $\alpha \in Con$ implies $op(\alpha) \in Con$
   for all $op \in \Sigma_1$
   $\alpha_1, \alpha_2 \in Con$ implies $op(\alpha_1, \alpha_2) \in Con$

1. $c?x.\langle t,\sigma\rangle \xrightarrow{c?v} \langle t,\sigma[v/x]\rangle$ for any $v \in Val$
   $c!v.\langle t,\sigma\rangle \xrightarrow{c![\![e]\!]} \langle t,\sigma\rangle$

2. $\alpha \xrightarrow{a} \alpha'$ implies $\alpha + \beta \xrightarrow{a} \alpha'$
   $\beta + \alpha \xrightarrow{a} \alpha'$

3. $\alpha \xrightarrow{a} \alpha'$ implies $\alpha \mid \beta \xrightarrow{a} \alpha' \mid \beta$
   $\beta \mid \alpha \xrightarrow{a} \beta \mid \alpha'$

4. $\alpha \xrightarrow{a} \alpha'$ implies $\alpha \setminus c \xrightarrow{a} \alpha' \setminus c$
   if $name(a) \neq c$

**Fig. 7.** Rules for $\xrightarrow{a}$

**Lemma 3.1.1.** For all $\phi \in$

To simplify reasoning about the operational behaviour we give an alternative characterisation of $\sqsubseteq_M$ in terms of the operational semantics of configurations. This is essentially the same as the alternative characterisation in [HI 89]. As there the actions are of the form $c?v$ or $c!v$ and when representing internal states using acceptance sets we need only remember the names of the channels along which an output can be sent or an input received but not the actual values themselves. As in §2.3 these will be called events.

$$Ev = \{c! \mid c \in Chan\} \cup \{c? \mid c \in Chan\}.$$

Now acceptance sets will be finite collections of finite sets of events. Note the difference to the acceptance sets in the definition of the Acceptance Trees. Here the acceptance sets are not necessarily saturated.

To define the alternative characterisation we need some notation, taken directly from [HI 89].

- For $s \in Act^*$ define $\alpha \stackrel{s}{\Longrightarrow} \beta$

of it to demonstrate how the existing proofs can be reused with configurations in the role of processes.

**Theorem 3.2.1.** For all $\alpha, \beta \in Con$

$\alpha \ll_M \beta$ if and only if

Then $\alpha \underline{must}\, b(s,B)$ where $B = \{b_1, \ldots, b_n\}$, but $\beta \not\underline{must}\, b(s,B)$ because of the unsuccessful computation

$$b(s,B)|\beta \rightarrowtail^* b(\varepsilon,B)|\gamma$$

where $\beta \stackrel{s}{\Longrightarrow} \gamma$ and $S(\gamma) = A$. □

As a corollary to the theorem we have the following:

**Corollary 3.2.1.** $\sqsubseteq_M$ over configurations is preserved by all the operations in $\Sigma$.

*Proof.* Similar to the proof for the corresponding result in [HI 89]. □

## 3.3. Full Abstractness

This last subsection is devoted to the proof of the full abstractness for configurations of the denotational model $AT^v$ with respect to the testing preorder $\sqsubseteq_M$. First we have to extend the definition of the interpretation of terms to that of configurations. We write $[\![\_]\!]$ instead of $AT^v[\![\_]\!]$.

**Definition 3.3.1.** The semantics of configurations is given as a function:

$$[\![\ ]\!] : Con \longrightarrow (Env_D \longrightarrow D)$$

defined by:

*i)* $[\![\langle t, \sigma\rangle]\!]\rho = [\![t]\!]\rho\sigma$

*ii)* $[\![op(\underline{\alpha})]\!]\rho = op([\![\underline{\alpha}]\!]\rho)$.

Also the definition of the finite approximations extends to configurations in the obvious way, and we can easily deduce that the meaning of a configuration is the limit of the meaning of its finite approximations:

$$[\![\alpha]\!] = \bigsqcup\{[\![\alpha^{(n)}]\!] \mid n \geq 1\}.$$

By full abstractness of the model we mean as usual

$$[\![\alpha]\!] \leq [\![\beta]\!] \quad \Leftrightarrow \quad \alpha \sqsubseteq_M \beta$$

for all configurations $\alpha, \beta$.

The proof follows very much the same lines as the corresponding one for the applicative case in [HI 89]. In the following we will outline the proof of full abstractness in [HI 89] and show how our new theory can be fitted into this existing proof which thus can be more or less reused. Recall that the denotational model is the same and that the configurations in the new settings play the role of processes in the previous one.

In [HI 89] we defined a transition relation and a divergence predicate in $AT^v$, and from that deduced an alternative characterization for the preorder $\leq$, called $\ll_M$. As we use the same model we can use the same definition. Thus we define the transition relation by

$$(\mathcal{A}, f) \stackrel{c*v}{\longrightarrow} T \text{ if } * \in \{!, ?\}, c* \in \bigcup \mathcal{A} \text{ and } f(c*)(v) = T$$

The divergence predicate, $\uparrow$, is defined by letting $\bot \uparrow$ and extending it in the usual way for $s \in Act^*$. $\downarrow s$ denotes the the negation of $\uparrow s$. The acceptance set of a tree after $s$, $\mathcal{A}(T, s)$ is defined by:

i) $\mathcal{A}(\bot$

$$
\begin{array}{rcl}
\langle (x := e); X, \sigma \rangle & = & \langle X, \sigma[v/x] \rangle \text{ where } v = [\![e]\!]\sigma \\
\langle be \rightarrow X_1, X_2, \sigma \rangle & = & \begin{cases} \langle X_1, \sigma \rangle & \text{if } [\![be]\!]\sigma = true \\ \langle X_2, \sigma \rangle & \text{if } [\![be]\!]\sigma = false \end{cases} \\
\langle op(\underline{X}), \sigma \rangle & = & op(\langle \underline{X}, \sigma \rangle) \\
\langle c?x.t, \sigma \rangle & = & c?x.\langle t, \sigma \rangle \\
\langle c!e.t, \sigma \rangle & = & c!v.\langle t, \sigma \rangle \text{ where } v = [\![e]\!]\sigma
\end{array}
$$

**Fig. 8.** Equations for Basic Configurations

**Lemma 3.3.1.** For all $\phi \in BCon$ there exists at most one $\alpha$ such that $\phi \succ\!\!\rightarrow \alpha$. For this $\alpha$, $\alpha$

$$\text{I}\quad \alpha \sqsubseteq \alpha \qquad\qquad \frac{\alpha \sqsubseteq \beta,\ \beta \sqsubseteq \gamma}{\alpha \sqsubseteq \gamma}$$

$$\text{II}\quad \frac{\alpha_i \sqsubseteq \beta_i}{op(\underline{\alpha}) \sqsubseteq op(\underline{\beta})} \quad \text{for every } op \in \sum$$

$$\text{III}\quad \frac{\alpha \sqsubseteq \beta}{c!e.\alpha \sqsubseteq c!e.\beta} \qquad \frac{\alpha[v/x] \sqsubseteq \beta[v/x] \text{ for every } v \in V}{c?x.\alpha \sqsubseteq c?x.\beta}$$

$$\text{IV}\quad \frac{\alpha \sqsubseteq \beta}{\alpha\rho \sqsubseteq \beta\rho} \qquad \frac{}{\alpha \sqsubseteq \beta} \quad \text{for every equation } \alpha \sqsubseteq \beta$$

$$\text{V}\quad \frac{[\![e]\!] = [\![e']\!]}{c!e.\alpha = c!e'.\alpha}$$

$$\text{VI}\quad \frac{}{c?x.\alpha = c?y.\alpha[y/x]} \quad \text{if } y \text{ does not occur free in } \alpha$$

**Fig. 9.** Proof System

**Corollary 3.3.1.** For all finite $\delta$

$$\delta \uparrow \ \Leftrightarrow\ \ \delta =_A \Omega.$$

*Proof.* By theorem 3.3.2 we may assume, that $\delta$ is in $wSF$. The result follows immediately from the structure of the weak sum forms and the strictness equations in Figure 2. □

Finally we introduce head normal forms for configurations. The definition is basically the same as the one given in [HI 89].

**Definition 3.3.3. (Head Normal Forms)** HNF is the least set which satisfies

1. $STOP \in HNF$
2. Let $\mathcal{A} \in sat(EV)$ and $f$ a partial function, which associates with every $e \in \bigcup\mathcal{A}$ of the form $c!$ a finite nonempty set, $f(c)$, of pairs of values and basic configurations. Then any configuration of the form

$$\sum\!\!\!\!\!\!\bigcirc\ \{\sum\{\alpha_{e,f} | e \in A\} | A \in \mathcal{A}\}$$

is in $HNF$, where

a) if $e$ is $c?$ then $\alpha_{e,f}$ is a configuration of the form $c?x.\langle t, \sigma\rangle$.

b) if $e$ is $c!$ then $\alpha_{e,f}$ is the configuration

$$\sum\{c!v.\phi_v | (v,\phi_v) \in f(c)\}$$

We have the following normalization theorem for convergent configurations:

**Theorem 3.3.3. (Normalization)** If $\alpha \downarrow$ then $\alpha =_A h(\alpha)$ for some $h(\alpha) \in HNF$.

*Proof.* Similar to the proof for Proposition 4.2.1 in [HI 89]. □

For the sake of completeness we state the above mentioned properties as a proposition:

**Proposition 3.3.1.** For all $\alpha \in Con$ and $s \in Act^*$

i)$[\![\alpha]\!] \downarrow s$ if and only if $\alpha \downarrow s$

ii) if $\alpha \downarrow s$ then $\mathcal{A}([\![\alpha]\!]$

## 4. Conclusion

In this paper we have represented a semantic theory for a process algebra which supports value-passing and is equipped with the imperative assignment construct. This is an direct extension of [HI 89], which handles an applicative version of the language, i.e. the same language but without assignment and stores.

As in the mentioned paper, the theory is approached from three different angles. Thus we define a denotational model for the language as a version of the model *Acceptance Trees*([He 88],[HI 89]). Further we give an axiomatization of the model and finally we define an operational behaviour in terms of testing. We show the equivalence of all three approaches.

Following the standard approach the semantic interpretation is given with respect to a store whereby we mean a total function which keeps track of the

$$\cup\{in_{AT}(c, \lambda v.F(t, g(c?)(v)) \mid c? \in B\}$$
$$\cup\{out_{AT}(c, v, F(f(c!)(v), u) \mid c! \in A,$$
$$\quad v \in domain(f(c!))\}$$
$$\cup\{out_{AT}(c, v, F(t, g(c!)(v)) \mid c! \in B,$$
$$\quad v \in domain(g(c!))\}$$

## References


[BHR 84] Brookes, S.D., Hoare, C.A.R. and Roscoe, A.W. "A Theory of Communicating Sequential Processes", *JACM* 31(7), 560-599 1984.

[Bri 86] Brinksma, E. "A Tutorial on LOTOS." *Proceedings of IFIP Workshop on Protocol Specification, Testing and Verification V*, M. Diaz, ed., pp. 73-84. North-Holland, Amsterdam, 1986.

[DNH 84] DeNicola, R. and M. Hennessy. "Testing Equivalences for Processes." *Theoretical Computer Science*, 24, 1984, pp. 83-113.

[FLP 84] Francez, N., Lehman, D. and Pnueli, A. "A Linear History of Semantics for Languages with Distributed Processing, *TCS, 32, 25-46*, 1984.

[Gue 81] Guessarian, I., "Algebraic Semantics", *Springer-Verlag Lecture Notes in Computer Science*, vol.99, 1981.

[HP 80]