INFORMATION AND COMPUTATION 107, 202–236 (1993)

A Theory of Communicating Processes with Value Passing

M. HENNESSY AND A. INGÓLFSDÓTTIR

School of Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton, Sussex BN1 9HQ, United Kingdom

A semantic theory of process algebras which allows processes to communicate values is described. A behavioural theory of testing is given for such processes and is modelled by an extension of Acceptance Trees. A proof system is also given for this model and is shown to be both sound and complete. Finally, the model is shown to be fully abstract with respect to the behavioural theory. © 1993 Academic Press, Inc.

1. INTRODUCTION

A semantic theory of pure processes has been presented in [He88]. By "pure processes" we mean processes whose only form of communication is pure synchronization: processes can only communicate by simultaneously performing uninterpreted synchronization actions or events. The theory has three different but complementary aspects which may most easily be explained as different methods for intepreting a range of process description languages or so-called process algebras. These enable one to give recursive definitions of processes in terms of a given set of combinators; each process algebra is characterized by a particular set of combinators. Those based on CCS [Mil80], assume a given set of uninterpreted actions Act, over which a complementation function, $\bar{}$, is defined. The simultaneous occurrence of an action, a, and its complement, $\bar{a}$, is considered to be a pure synchronization. A large number of combinators for processes have been suggested. These considered in [He88] include external nondeterminism, $+$, internal nondeterminism, $\oplus$, the parallel operator, $|$, the action restriction operator, $\backslash c$, and action renaming, $[R]$.

The first interpretation of this language is behavioural and consists of two parts, an operationl semantics and a theory of testing based on this operational semantics. This leads to a behavioural equivalence on processes, $p \approx_M q$, meaning that both p and q pass exactly the same tests. The second interpretation is denotational; a denotational model for the language is given, called Acceptance Trees. Briefly, these are finite-

0890-5401/93 $5.00

branching trees where each branch is labelled by an action or event. They are deterministic in that from each node there is at most one branch labelled with a given event. Nondeterminism is modelled by labelling the nodes with Acceptance sets of events. These are finite sets of events and each individual set models a possible internal state of the process being interpreted. The third and final interpretation is equational or axiomatic. A set of equations over the combinators is presented and they, together with a powerful form of induction, give rise to a transformational proof system for deriving identities between processes. For pure processes all of these three different interpretations coincide; i.e., two processes are behaviourally equivalent if and only if they are interpreted as the same Acceptance Tree if and only if they can be proved equal.

The aim of this paper is to extend this theory to processes which not only may synchronize with each other but also may transfer data between processes during synchronization. The extended language we consider may be viewed as a decription language for communicating agents which send values to each other over virtual communication channels. In this setting the set of actions Act is now interpreted; each action represents either the sending of a value v to a channel c, $c!v$, or the reception of a value v from a channel c, $c?v$. Moreover, these actions are complementary to each other so that a synchronization now consists of the simultaneous transmission and reception of a value v along a channel c. Whereas in the pure language one may define processes such as

$$P = a.(b.P + c.P),$$

where a, b, and c are primitive actions, in the extended language we can define processes such as

$$P = in?x.\mathit{If}\ \mathit{Even}(x)\ \mathit{Then}\ \mathit{out}1!x.P$$
$$\mathit{Else}\ \mathit{out}2!x.P,$$

where in, $out1$, and $out2$ are the names of the channels. In addition, we have a typical collection of process combinators such as $|$, $+$, $\oplus$, and $\backslash a$, with the result that the language being investigated is quite powerful.

The behavioural interpretation of this extended language is straightforward. The operational semantics of full-CCS [Mil89] is suitably modified for our setting and using the general definitions of testing from [He88] we obtain a behavioural equivalence, or, more generally, a preorder. A denotational interpretation for the extended language may also be given by extending the idea of Acceptance Trees. The model we present is a minor variation on that defined originally by Milne in [Miln88]. An Acceptance Tree is still a finite-branching tree where each branch is

labelled deterministically by an event. For pure processes an event is simply an element of *Act*, the predefined set of actions. In the extended setting there are two kinds of events, input events of the form $c?$ and output events of the form $c!$, where c is a channel name. Nondeterminism is once more modelled by labelling the nodes with Acceptance sets of events. However, there is a subtlety. In the pure case a branch in an Acceptance Tree always leads to another Acceptance Tree, a subtree. Now events have more complicated sequels. If a branch is labelled by an output event its sequel is a finite partial function from values to Acceptance trees. On the other hand, the sequel to an input event is a function, associating with each possible value an Acceptance Tree. The result is a well-behaved algebraic cpo in which our extended language may be interpreted. We prove that every finite element in this model is definable and that two processes are behaviourally equivalent if and only if they are interpreted as the same Acceptance Tree.

The final interpretation of the language is obtained by augmenting the original proof system. Most of the original equations remain unchanged and a few are generalized in an obvious manner. For example, the original equation

$$a.X \oplus a.Y = a.(X \oplus Y)$$

is replaced by the pair

$$c?x.X \oplus c?x.Y = c?x.(X \oplus Y)$$
$$c!e.X \oplus c!c.Y = c!e.(X \oplus Y).$$

The equation

$$a.X + a.Y = a.X \oplus a.Y$$

is also generalized in a similar manner; but there is an extra ingredient. For input guards we obtain

$$c?x.X + c?x.Y = c?x.X \oplus c?x.Y$$

but for output guards we obtain the more general

$$c!e.X + c!e'.Y = c!e.X \oplus c!e'.Y.$$

Here the output channels must be the same but e and e' may be different expressions. We also need some obvious equations for boolean values such as

$$\textit{If } T \textit{ Then } X \textit{ Else } Y = X.$$

In addition we need a new form of induction because the collection of values *Val* may be infinite. We use the infinitary rule:

$$\frac{p[v/x] = q[v/x], \text{ for every } v \text{ in } Val}{c?x.p = c?x.q}$$

With these new rules (and some further obvious additions) we may once more prove that two processes are behaviourally equivalent if and only if they can be proved equal in the extended proof system.

In Section 2 we describe the language and its behavioural theory. This consists of a structural operational semantics, in Section 2.2, and a theory of testing based on it, in Section 2.3. In this section we also give an alternative characterization of the resulting preorder on prcesses. In Section 3 we describe the denotational semantics. The first section is devoted to the definition of general models suitable to our language. These are called *natural interpretations*. In the next section, 3.2, we show how to generalize the model AT_s from [He88] to take value-passing into account. This model was originally suggested in [Miln88]. In Section 3.3 we show that the extended model, called AT_s^v, is a natural interpretation, and develop some of its properties. In Section 4.1 we describe the proof system for the language and show that it is sound with respect to the model AT_s^v. In the next section we show that it is also complete. Finally, in Section 4.3, we show that the model AT_s^v is fully abstract with respect to testing; i.e., it identifies and only identifies processes which cannot be distinguished via testing. The paper concludes with a comparison with other approaches to modeling value-passing. In particular we show why the standard model AT_s is not adequate for this purpose.

2. The Language

In this section we give the syntax for the language and its operational semantics. It is essentially full CCS [Mil80], but with τ replaced by internal nondeterminism, $\oplus$, although the theory to be presented may also be used to explain other forms of parallelism such as that used in CSP [Hoa85], and LOTOS [Bri86]. In Section 2.3 we quickly present the theory of testing as it applies to our language. We also give a generalization of the alternative characterization from [He88] and state further properties of the preorder which will be required in later sections.

2.1. *The Syntax*

The language we consider is rather abstract as we wish to concentrate on its parallel aspects. For this reason we simply assume some predefined

syntactic category of expressions, *Exp*, ranged over by e. This should include, at least, a set of variable symbols, *Var*, ranged over by x, and a nonempty countable set of value symbols, *Val*, ranged over by v. As an example the reader might keep in mind the case where *Val* is the set of numerals, N, and *Exp* is obtained from N and *Var* using operators such as *Succ*, *Pred*, etc. We also assume a notion of closed expression, one containing no occurrences of variables, and substitution, $e[e'/x]$, a new expression which is obtained from e by substituting the expression e' for all occurrences of the variable x. We also assume a separate syntactic category of boolean expressions *BExp*, ranged over by be. Here the set of boolean variables is denoted by *BVar* and ranged over by bx and the only values are the constants T and F. Of course we expect the language for boolean expressions to use that for expressions and one crucial result, Proposition 2.8, depends on the expressive power of *BExp* with respect to the set of values *Val*.

The set of allowed operators is *NIL*, Ω of arity 0, $\backslash c$, where c ranges over a predefined set of channel names, *Chan*, and $[R]$, where R ranges over *renamings*, i.e., finite permutations of *Chan*, all of arity one, and $\oplus$, $+$, and $|$ of arity two. We use Σ to denote this collection of operators and Σ_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 (process) terms is then defined by the BNF-definition

$$t ::= op(t_1, \ldots, t_k),\ op \in \Sigma_k \mid P \mid pre.t \mid recP.t$$
$$\mid \mathit{If}\ be\ \mathit{Then}\ t\ \mathit{Else}\ t$$
$$pre ::= c!e \mid c?x.$$

The construction $recP.\_$ binds occurrences of process names which gives rise in the usual way to free and bound names and to closed and open terms. We use $t[u/P]$ to denote the term which results from substituting the term u for every free occurrence of P in t. We will sometimes use a more general form of substitution. If θ is a mapping from *PN* to the set of terms then $t\theta$ is the term which results from simultaneously substituting $\theta(P)$ for each free occurrence of P in t. Value-variables may also be bound in process terms with construction $c?x.\_$, giving rise to free and bound variables. Substitution of value-expressions for value variables is also extended to process terms in the obvious way: $t[e/x]$ is the process term which results from substituting the value-expression e for every free occurrence of the value-variable x in the process term t.

DEFINITION 2.1. Let VPL (*Value Passing Language*) denote the set of all process terms which contain no free occurrences of value-variables. By

a *closed* term in VPL we mean a term in VPL which contains no free process names. These will often be referred to as *processes*. The set of processes will be denoted by *Proc* and ranged over by p, q, r, etc.

Note that the process names are not parameterized by value-expressions. Adding this feature would undoubtably give a more usuable language but for convenience we omit it. However, all for our results could easily be extended to this more powerful language.

A finite term is one that contains no occurrences of *rec* . . Every term determines a set of finite terms, its *finite approximations*, $App(t)$, in the usual way:

DEFINITION 2.2. For any term t the nth finite approximation, t^n, is defined inductively as:

(i) $t^0 = \Omega$

(ii)(a) $(op(\underline{t}))^{n+1} = op(\underline{t}^{n+1})$

(b) $(pre.t)^{n+1} = pre.t^{n+1}$

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

(d) $(If\ be\ Then\ t\ Else\ u)^{n+1} = If\ be\ Then\ t^{n+1}\ Else\ u^{n+1}$.

The semantic theory we develop is such that the meaning of every process p is completely determined by that of its finite approximations, $\{p^n, n \geqslant 0\}$.

We end this section with some examples. When writing terms in VPL we use the usual syntactic conventions of CCS and assign the usual precedences to the operators.

EXAMPLE 2.3. Consider the process

$$recP.in?x.If\ x = 0\ Then\ c?y.out!y.NIL\ Else\ c?y.(in!(x-1).NIL \mid P)\backslash in.$$

It uses three channels, *in*, *out*, and *c*. It first inputs a number, n, from channel *in* and outputs along channel *out* the nth value received along channel *c*.

EXAMPLE 2.4. The process

$$recX.in?x.left!x.((in!(x+1).NIL \mid X)\backslash in)$$

uses two channels, *in* and *left*. It first inputs a number, n, from *in* and subsequently outputs on *left* the infinite sequence of values $n, n+1, n+2, \ldots$.

2.2. *Operational Semantics*

The language VPL is given an operational semantics in the usual way using labelled transition systems. We define an extended labelled transition system $\langle Proc, Act, \rightarrow, \rightarrowtail \rangle$, where

- Act is a set of actions
- $\rightarrow \subseteq Proc \times Act \times Proc$
- $\rightarrowtail \subseteq Proc \times Proc$.

$(p, a, q) \in \rightarrow$ is usually written $p \xrightarrow{a} q$ and intuitively it means that the process p may perform the action a and thereby be transformed into q. $p \rightarrowtail q$ may be read as "p may evolve spontaneously to q."

The set Act consists of all input events of the form $c?v$ and all output events of the form $c!c$ where $c \in Chan$ and $v \in Val$. The relations $\xrightarrow{a}$ and $\rightarrowtail$ are defined to be the least relations which satisfy the rules given in Figs. 1 and 2. These rules presuppose an evaluation mechanism for closed expressions: $[\![c]\!]$ gives the value in Val of the expression c and $[\![bc]\!]$ returns either T or F. In a less abstract language we would have to elaborate on these evaluation mechanisms. The rule for communication, Rule 2, Fig. 1, uses a complementation notation for actions: $\overline{c?v}$ is $c!v$ and $\overline{c!v}$ is $c?v$. The rules for channel hiding also use the obvious notation $name(a) \neq c$ to indicate that the action a does not use the channel c. In the rule for renaming $R(a)$ denotes the action obtained by replacing the channel in a by $R(name(a))$.

The rules themselves are quite straightforward. Those for boolean expressions and input/output are taken from [Mil80] and the remainder are directly from [He88]. The resulting labelled transition system is not in

1.	$c?x.p \xrightarrow{c?v} p[v/x]$ $c!e.p \xrightarrow{c![\![e]\!]} p$	for any $v \in Val$
2.	$p \xrightarrow{a} p'$ implies	$p + q \xrightarrow{a} p'$ $q + p \xrightarrow{a} p'$
3.	$p \xrightarrow{a} p'$ implies	$p \mid q \xrightarrow{a} p' \mid q$ $q \mid p \xrightarrow{a} q \mid p'$
4.	$p \xrightarrow{a} p'$ implies	$p \backslash c \xrightarrow{a} p' \backslash c$ if $name(a) \neq c$
5.	$p \xrightarrow{a} p'$ implies	$p[R] \xrightarrow{R(a)} p'[R]$
6.	$[\![be]\!] = T$, $p \xrightarrow{a} p'$ implies *If be Then p Else q* $\xrightarrow{a} p'$ $[\![be]\!] = F$, $q \xrightarrow{a} q'$ implies *If be Then p Else q* $\xrightarrow{a} q'$	

FIG. 1. Rules for $\xrightarrow{a}$.

1. $p \xrightarrow{a} p', q \xrightarrow{\bar{a}} q'$ implies $p \mid q \rightarrowtail p' \mid q'$
2. $p \oplus q \rightarrowtail p$
   $p \oplus q \rightarrowtail q$
3. $\Omega \rightarrowtail \Omega$
4. $recP.t \rightarrowtail t[recP.t/P]$
5. $p_i \rightarrowtail p_i'$ implies $op(\cdots, p_i, \cdots) \rightarrowtail op(\cdots, p_i', \cdots)$ for $op \in \{+, |, \backslash c, [R]\}$
6. $[\![be]\!] = T$, $p \rightarrowtail p'$ implies *If be Then p Else q* $\rightarrowtail p'$
   $[\![be]\!] = F$, $q \rightarrowtail q'$ implies *If be Then p Else q* $\rightarrowtail q'$

FIG. 2. Rules for $\rightarrowtail$.

general finite branching, at least if the set of values *Val* is infinite. In this case any process $c?x.p$ will have an infinite number of derivations since for every $v \in Val$ $c?x.p \xrightarrow{c?v} p[v/x]$. However, certain aspects of the transition system are finitary. First some notation:

$$InC(p) = \{c \mid \exists p', v.\, p \xrightarrow{c?v} p'\}$$
$$OutD(p) = \{p' \mid \exists c, v.\, p \xrightarrow{c!v} p'\}$$
$$IntD(p) = \{p' \mid p \rightarrowtail p'\}$$
$$D(p, a) = \{p' \mid p \xrightarrow{a} p'\}.$$

THEOREM 2.5. *For every p in VPL, InC(p), OutD(p), IntD(p) and D(p, a) are finite.*

Proof. By structural induction on p. Note that the case $recP.t$ is trivial since $IntD(recP.t) = \{t[recP.t/P]\}$ and $InC(recP.t) = OutD(recP.t) = \varnothing$. Indeed the only nontrivial case is when p has the form $p \mid q$. In this case we show that $IntD(p)$ is finite, leaving the rest to be checked by the reader. If $q \mid r \rightarrowtail p'$ then there are three possibilities:

(i) p' is $q \mid r'$ where $r \rightarrowtail r'$
(ii) p' is $q' \mid r$ where $q \rightarrowtail q'$
(iii) p' is $q' \mid r'$ where $q \xrightarrow{a} q'$ and $r \xrightarrow{\bar{a}} r'$ for some action a.

By induction cases (i) and (ii) only give rise to a finite number of possibilities. In the third case either a or $\bar{a}$ must be an output action. Since $OutD(q)$ and $OutD(r)$ are finite, by induction there are also a finite number of $q' \mid r'$ in this case. ∎

2.3. *Testing*

In this section we apply the general theory of testing from [He88] to VPL, concentrating on the *MUST* case. It is straightforward and holds no surprises.

A *test* is any process from VPL which may use in addition to the channels in *Chan* a special channel w for reporting success. We say p *must* e, where p is a process and e is a test, if in every complete computation

$$p \mid e = p_0 \mid e_0 \rightarrowtail p_1 \mid e_1 \cdots p_k \mid e_k \rightarrowtail \cdots$$

there exists some $n \geqslant 0$ such that $e_n \xrightarrow{w!v}$ for some value v. By a complete computation we mean a maximal sequence of derivations; i.e., either it is infinite, or if it is finite and $p_n \mid e_n$ is the finite element then $p_n \mid e_n \rightarrowtail q$ for no q. Then $p \sqsubseteq_{\mathrm{M}} q$ if for every test e,

$$p \text{ must } e \text{ implies } q \text{ must } e.$$

This relation is extended to arbitrary terms in VPL by letting $t \sqsubseteq_{\mathrm{M}} u$ if for every closed substitution, θ, i.e., a mapping from PN to $Proc$, $t\theta \sqsubseteq_{\mathrm{M}} u\theta$. We take the preorder $\sqsubseteq_{\mathrm{M}}$ and its kernel $\approx_{\mathrm{M}}$ to represent the primary semantic theory for our language and in later sections we give denotational and equational characterizations of them. At this point it is not straightforward to show that $\sqsubseteq_{\mathrm{M}}$ is a well-behaved relation; for example that

- it is preserved by all the constructs in the language
- for any test e, p *must* e if and only if p^n *must* e for some finite approximation p^n.

To prove results such as these we need more technical machinery. One useful fact is an alternative characterization of $\sqsubseteq_{\mathrm{M}}$ in terms of the operational semantics of processes. This is essentially the same as the alternative characterization in [He88] with one change. There Acceptance sets are used; these are finite collection of finite sets of actions, each finite set intuitively representing the possible internal states. Here actions are of the form $c?v$ or $c!v$. However, 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. 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.

To define the alternative characterization we need some notation:

- For $s \in Act^*$ define $p \stackrel{s}{\Longrightarrow} q$ by
  (i) $p \stackrel{\varepsilon}{\Longrightarrow} q$ if $p \rightarrowtail^* q$
  (ii) $p \stackrel{as'}{\Longrightarrow} q$ if for some $p', p'', p \stackrel{\varepsilon}{\Longrightarrow} p', p' \stackrel{a}{\longrightarrow} p''$ and $p'' \stackrel{s'}{\Longrightarrow} q$.
- $L(p) = \{s \mid \text{for some } q,\ p \stackrel{s}{\Longrightarrow} q\}$.
- Define $\downarrow$, $\downarrow s$, $\uparrow$ and $\uparrow s$ by
  (i) $p\downarrow$ if there is no infinite internal computation

$$p = p_0 \rightarrowtail p_1 \rightarrowtail \cdots$$

  (ii) $p \downarrow \varepsilon$ if $p\downarrow$
  (iii) $p \downarrow as'$ if $p\downarrow$ and if $p \stackrel{a}{\Longrightarrow} p'$ then $p' \downarrow s'$
  (iv) $p\uparrow$ if $p\downarrow$ is false and $p \uparrow s$ if $p \downarrow s$ is false.
- Define $S(p) \subseteq Ev$ by

$$S(p) = \{c? \mid \text{for some } v,\ p \stackrel{c?v}{\Longrightarrow}\} \cup \{c! \mid \text{for some } v,\ p \stackrel{c!v}{\Longrightarrow}\}.$$

- Define $\mathscr{A}(p, s)$, the Acceptance set of events of p after s, by

$$\mathscr{A}(p, s) = \{S(p') \mid p \stackrel{s}{\Longrightarrow} p'\}.$$

DEFINITION 2.6. For $p, q \in \mathrm{VPL}$, $p \ll_{\mathrm{M}} q$ if for every $s \in Act^*$

$$p \downarrow s \Rightarrow \text{(a) } q \downarrow s$$
$$\text{(b) for every } A \in \mathscr{A}(q, s) \text{ there is some } B \in \mathscr{A}(p, s) \text{ such that } B \subseteq A.$$

Condition (b) will often be abbreviated as $\mathscr{A}(q, s) \Subset \mathscr{A}(p, s)$.

In the remainder of this section we show that

$$p \ll_{\mathrm{M}} q \text{ if and only if } p \sqsubseteq_{\mathrm{M}} q. \tag{1}$$

This characterization makes the analysis of $\sqsubseteq_{\mathrm{M}}$ much more straightforward. For example, we have the following corollary.

COROLLARY 2.7. *$\sqsubseteq_{\mathrm{M}}$ is preserved by all the operations in Σ.*

Proof. In view of (1) it is sufficient to check the result for $\ll_{\mathrm{M}}$, which is straightforward but tedious. ∎

The proof of (1) follows closely the proof of the corresponding result in [He88]. We omit the proof that $p \ll_{\mathrm{M}} q$ implies $p \sqsubseteq_{\mathrm{M}} q$ as the interested

reader may easily construct it from Theorem 2.2.12 on p. 73 of [He88]. Instead we give the proof of

PROPOSITION 2.8. *$p \sqsubseteq_M q$ implies $p \ll_M q$.*

Proof. For each $s \in Act^*$ and $a \in Act$ define the tests $con(s)$ and $rej(s, a)$ as follows:

(i) $con(\varepsilon) = w \oplus w$
$con(c?v.s) = c!v.con(s) + (w \oplus w)$
$con(c!v.s) = (c?x.\, If\ x = v\ Then\ con(s)\ Else\ w \oplus w) + (w \oplus w)$

(ii) $rej(\varepsilon, c?v) = c!v.NIL + (w \oplus w)$
$rej(\varepsilon.c!v) = (c?x.\, If\ x = v\ Then\ NIL\ Else\ w \oplus w) + (w \oplus w)$
$rej((c?v)s, a) = c!v.rej(s.a) + (w \oplus w)$
$rej((c!v)s, a) = (c?x.\, If\ x = v\ Then\ rej(s, a)\ Else\ w \oplus w) + (w \oplus w).$

With these definitions one can show that

$$p\ \underline{must}\ con(s) \text{ if and only if } p \downarrow s$$

and

$$\text{if } p \downarrow s \text{ then } p\ \underline{must}\ rej(s, a) \text{ if and only if } sa \notin L(p).$$

As a corollary we have

$$\text{if } p \sqsubseteq_M q \text{ and } p \downarrow s \text{ then } q \downarrow s$$

and

$$\text{if } p \sqsubseteq_M q \text{ then } p \downarrow s \text{ and } s \in L(q) \text{ implies } s \in L(p). \tag{2}$$

Now suppose $p \sqsubseteq_M q$. We show that $p \ll_M q$. Assume that $p \downarrow s$. This implies $q \downarrow s$. Let $A \in \mathscr{A}(q, s)$. Because of (2) $\mathscr{A}(p, s)$ is not empty. Also since $p \downarrow s$ it follows from Theorem 2.5 that $\mathscr{A}(p, s)$ is finite, say $\{B_1, ..., B_n\}$. We have to show, that $B_i \subseteq A$ for some i. Assume that this is not true. This means $B_i \backslash A \neq \varnothing$ for all i and we can choose $b_i \in B_i \backslash A$ for $i = 1, ..., n$. Now for $e \in Ev$ and $L \subseteq Ev$ let $ac(e)$ be defined by

$$ac(c?) = c!0.w!0.NIL$$

$$ac(c!) = c?x.w!0.NIL$$

and $ac(s, L)$ by

$$ac(\varepsilon, L) = \Sigma\{ac(a) \mid a \in L\}$$

$$ac((c?v)s, L) = c!v.ac(s, L) + (w \oplus w)$$

$$ac((c!v)s, L) = (c?x.\, If\ x = v\ Then\ ac(s, L)\ Else\ w!0.NIL) + (w \oplus w).$$

Then p *must* $ac(s, B)$, where $B = \{b_1, ..., b_n\}$, but q *must* $ac(s, B)$ because of the unsuccessful computation

$$ac(s, B) \mid q \rightarrowtail^* ac(\varepsilon, B) \mid r,$$

where $q \stackrel{s}{\Longrightarrow} r$ and $S(r) \subseteq A$. ∎

Note that this characterization of $\sqsubseteq_M$, which is crucial for the remainder of the paper, depends on the ability to test for equality between elements of *Val*.

3. Denotational Semantics

In this section we give a mathematical model for the language VPL. The section is divided into three subsections. In the first subsection we describe a general class of mathematical models for the language. This consists of a natural generalization of the idea of Σ-*cpo*. The generalization is required in order to interpret the input and output operations which cannot be accommodated within the framework of Σ-*cpos*. Following [HP80] these more general structures are called *natural interpretations*. In the next two subsections we describe a particular natural interpretation called *Strong v-Acceptance Trees*, AT_s^v. They are a natural extension of the Strong Acceptance Trees, AT_s, in [He88]. In Section 3.2 the original domain AT_s is reformulated and this reformulation facilitates its extension to a domain suitable for interpreting processes with value-passing, AT_s^v. In Section 3.3 we show that this extension is a natural interpretation by defining the appropriate functions over it. Throughout this section we assume that the reader is familiar with denotational and algebraic semantics. Details may be found in [Smt86, He88, Gue81].

3.1. *Natural Interpretations*

VPL consists of recursive terms constructed using the operator symbols Σ and two forms of prefixing, $c?x$ and $c!e$. To interpret the standard operators in Σ, we use a Σ-*cpo* which consists of a carrier, D, a *cpo*, and for each symbol op in Σ a continuous function op_D over D of the appropriate arity. However, the prefixing operations cannot be interpreted in a similar manner. For example, $c?x$ does not take a process and return a process; it is a binding operator which takes an open term and returns a process, at least if x is the only free value variable in its argument. Semantically the operation of *input* has the type

$$Chan \times (Val \rightarrow D) \rightarrow D,$$

where D is the proposed interpretation of processes. Similarly *output* has the type

$$Chan \times Val \times D \to D.$$

Such general structures, consisting of a Σ-*cpo* and *input* and *output* functions of these types, have previously been used in [HP80], where they are called natural interpretations. We adopt this terminology.

DEFINITION 3.1. A *natural interpretation* (for the language VPL) consists of a triple, $\langle D, out_D, in_D \rangle$, where

(i) D is a Σ-*cpo*

(ii) $out_D : (Chan \times Val \times D \to D)$ is a total function continuous in its third argument.

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

Given such a natural interpretation, which we informally refer to as D, we can define a semantics of VPL following the usual approach of denotational semantics. Recall that terms in VPL contain no free value variables but they may contain free process names. So the semantic mapping only requires as a parameter environments for interpreting the latter. We let Env_D be the set of D-environments, i.e., mappings from PN to D, ranged over by ρ. Then the semantics is given as a function,

$$D[\![\ ]\!] : \text{VPL} \to [Env_D \to D],$$

and is defined by structural induction on VPL:

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

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

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

(iv) $D[\![If\ be\ Then\ t\ Else\ u]\!]\rho = D[\![t]\!]\rho$ if $[\![be]\!] = T$
$D[\![If\ be\ Then\ t\ Else\ u]\!]\rho = D[\![u]\!]\rho$ if $[\![be]\!] = F$

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

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

This is only a mild generalization of the usual algebraic semantics and most of the usual results also remain true. For example, the meaning of a term is the limit of the meaning of its finite approximations:

$$D[\![t]\!]\rho = \bigsqcup \{D[\![t^n]\!]\rho \mid n \geqslant 0\}.$$

We wish to construct a particular natural interpretation which properly refects the testing preorder $\sqsubseteq_M$. This is the subject of the next subsection.

3.2. *Strong Acceptance Trees*

We start with a brief review of the Strong Acceptance Trees, AT_s, defined in [He88]. We then give an alternative description which faciliatates the generalization to value-passing.

The definition presupposes a set of actions *Act*. Then a Strong Acceptance Tree is a finite-branching tree in which the branches are labelled by actions and the nodes are noted either as *open*, denoting a divergent computation, or closed, in which case they are labelled by an Acceptance set, i.e., a finite nonempty collection of finite subsets of *Act*; intuitively each Acceptance set represents a possible internal state. An example may be found in Fig. 3. There are some simple conditions on these trees:

(i) Determinancy: from any node there is at most one successor branch from that node labelled by a particular action.

(ii) The Acceptance sets are saturated: they satisfy

(a) $A \in \mathscr{A}$, $B \in \mathscr{A}$ implies $A \cup B \in \mathscr{A}$

(b) $A \in \mathscr{A}$, $B \in \mathscr{A}$, $A \subseteq C \subseteq B$ implies $C \in \mathscr{A}$.

(iii) If a node is open it has no successors.

(iv) Acceptance sets contain only actions from successor branches: let $Act(\mathscr{A}) = \bigcup \{A \mid A \in \mathscr{A}\}$ and $Succ(n)$ denote the set of actions labelling successor branches of the node n. Then if $\mathscr{A}$ labels node n, $Succ(n) = Act(\mathscr{A})$.

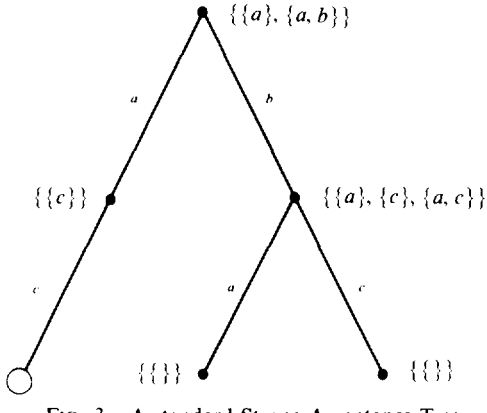

FIG. 3. A standard Strong Acceptance Tree.

The first condition is there because nondeterminism is reflected in the Acceptance sets. The second and third are required in order to properly reflect the testing preorder. The fourth is a natural consequence.

Each such tree is entirely determined by the partial function, which given an action returns the *sequel* with respect to this action, i.e., the subtree obtained by following the branch labelled by the action. The generalization of Acceptance Trees to hand value-passing processes is obtained by modifying the possible sequels and replacing Act by a more general set of events. Let Ev be an arbitrary set of events and $SEQUELS$ an arbitrary *cpo* whose elements we refer to as "sequels." We use $\mathscr{A}(Ev)$ to denote the collection of saturated Acceptance sets over Ev and for $\mathscr{A} \in \mathscr{A}(Ev)$ we use $Ev(\mathscr{A})$ to denote $\bigcup \mathscr{A}$. Also $A \rightharpoonup B$ denotes the set of partial functions from A to B and $Fin(A \rightharpoonup B)$ the set of partial functions with a finite but nonempty domain.

Given such a set Ev and $SEQUELS$ let

$$H(Ev, SEQUELS) = \{(\mathscr{A}, f) \mid \mathscr{A} \in \mathscr{A}(Ev), \; f \in Ev \rightharpoonup SEQUELS, \text{ and } domain(f) = Ev(\mathscr{A})\}.$$

Elements of $H(Ev, SEQUELS)$ can be ordered by

$$(\mathscr{A}, f) \leqslant (\mathscr{B}, g) \text{ if (i) } \mathscr{B} \subseteq \mathscr{A} \text{ and (ii) } f \leqslant g.$$

In the second condition the partial functions f and g are ordered in a nonstandard fashion: $f \leqslant g$ if $domain(g) \subseteq domain(f)$ and for every $e \in domain(g)$, $f(e) \leqslant g(e)$. Here condition (i) ensures that $domain(g) \subseteq domain(f)$ but this ordering on partial functions will also be used in situations where this is not automatically the case. This makes $H(Ev, SEQUELS)$ into a partial order and all directed sets have limits. But it lacks a least element. Let $H(Ev, SEQUELS)_\perp$ denote the result of adding a bottom element, with the ordering extended in the natural way. Then $H(Ev, SEQUELS)_\perp$ is a *cpo*. The standard Strong Acceptance Trees with respect to a set of actions Act can now be obtained as the least solution to the equations

$$Ev = Act \quad \text{(as sets)}$$
$$AT_s = H(Ev, SEQUELS)_\perp$$
$$SEQUELS = AT_s.$$

To ensure that a solution exists we must extend $H(Ev, \_)$ to a functor in the category of *cpos* with embeddings, $\underline{\underline{CPO}}$ (see [Plo81, SP82]), and ensure that it is continuous. However, this is straightforward.

To model VPL we need more general events and more general sequels. We have already seen that two kinds of events are necessary:

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

Unfortunately the different kinds of events require different kinds of sequels, to allow for which we need to introduce a somewhat clumsy notation. Let $\uplus$ denote disjoint union of sets. Then $(A_1 \uplus A_2 \rightharpoonup B_1 \uplus B_2)$ is used to denote the set of all type respecting partial functions f from $A_1 \uplus A_2$ to $B_1 \uplus B_2$, i.e., functions with the property that $a \in A_i$ implies $f(a) \in B_i$ whenever $f(a)$ is defined. Now let AT_s^v denote the least solution in $\underline{\underline{CPO}}$ of

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

$$AT_s^v = H(Ev, SEQUELS)_\perp$$

$$SEQUELS = (Val \to AT_s^v) \uplus Fin(Val \rightharpoonup AT_s^v).$$

Note that here we now expect the functions in H to be type respecting. So the sequel to an event $c?$ is a total function from Val to AT_s^v whereas that of the event $c!$ is a finite partial function from Val to AT_s^v. Intuitively if f is the sequel to $c?$ then for every value v $f(v)$ describes the behaviour of the process after it receives v along channel c. If f is the sequel to $c!$ then for every v in its domain $f(v)$ describes the behaviour of the process after it transmits v along the channel c. In this case f is a finite partial function since intuitively in any state a process may only transmit a finite number of values along the channel. Here the nonstandard definition of the ordering on partial functions play a significant role. Intuitively it means that a process can be "improved" in the ordering by restricting the set of values it may send along a particular channel.

THEOREM 3.2. *AT_s^v is an ω-algebraic cpo.*

Proof. The equation above is solved in the category $\underline{\underline{CPO}}$. F, defined by

$$F: X \to H(Ev, (Val \to X) \uplus Fin(Val \rightharpoonup X))_\perp,$$

is transformed into a functor in the obvious way. Since it is continuous functor AT_s^v at least exists. Moreover if D is an ω-algebraic *cpo* then, because Val is countable, $Val \to D$ and $Fin(Val \rightharpoonup D)$ are ω-algebraic as well and therefore so is $F(D)$. It follows that F preserves ω-algebraicness and so AT_s^v is ω-algebraic. (See [Plo81, Sect. 6].) ∎

It turns out that we can reformulate the partial order on AT_s^v so that it is very similar in spirit to the preorder $\ll_M$ on processes, which is the basis of the alternative characterization of $\sqsubseteq_M$. This reformulation will be used in Section 4 to prove the full-abstractness result. As with standard Acceptance Trees, elements of AT_s^v may be characterized by sequences of actions and Acceptance sets. To define these we use the canonical isomorphism pair

$$fold : F(AT_s^v) \mapsto AT_s^v$$
$$unfold : AT_s^v \mapsto F(AT_s^v).$$

This enables us to introduce the useful notion of an a-successor to a tree in AT_s^v. For each action a in Act we define an infix partial function $\xrightarrow{a}$ by

$$T \xrightarrow{a} T' \text{ if (i) } a \text{ is } c!v,\ unfold(T) = (\mathscr{A}, f), \text{ and } T' \text{ is } f(c!)(v)$$
$$\text{or (ii) } a \text{ is } c?v,\ unfold(T) = (\mathscr{A}, f), \text{ and } T' \text{ is } f(c?)(v).$$

We can now define $\mathscr{A}(T, s)$, the Acceptance set of T after s, by

$$\text{(i)}\quad \mathscr{A}(T, \varepsilon) = \begin{cases} A & \text{if } unfold(T) = (\mathscr{A}, f) \\ \varnothing & \text{otherwise, i.e., } unfold(T) = \perp \end{cases}$$

$$\text{(ii)}\quad \mathscr{A}(T, as) = \begin{cases} \mathscr{A}(T', s) & \text{if } T \xrightarrow{a} T' \\ \varnothing & \text{otherwise, i.e., } unfold(T) = \perp. \end{cases}$$

It will also be useful to define convergence predicates over trees, similar to those defined over processes. Let $\downarrow s$ be the predicate over AT_s^v defined by

(i) $T \downarrow \varepsilon$ if $T \neq \perp$

(ii) $T \downarrow as$ if $T \downarrow \varepsilon$ and $T \xrightarrow{a} T'$ implies $T \downarrow s$.

Then it is easy to check that $T \downarrow s$ and $s \in L(T)$ implies $\mathscr{A}(T, s)$ is not empty.

In analogy with the alternative characterization of $\sqsubseteq_M$ in the previous section we may now use the acceptance sets of trees to define an ordering, also called $\ll_M$, over AT_s^v.

DEFINITION 3.3. For $T, U \in AT_s^v$ let $T \ll_M U$ if for every $s \in Act^*$,

$$T \downarrow s \Rightarrow \text{(i)}\quad U \downarrow s$$
$$\text{(ii)}\quad \mathscr{A}(U, s) \subseteq \mathscr{A}(T, s).$$

Note that $T \ll_M U$ implies that if $T \downarrow a$ and $U \xrightarrow{a} U'$ then $T \xrightarrow{a} T'$ and $T' \ll_M U'$. This property is useful in the following theorem.

THEOREM 3.4. *For T, U in AT_s^v, $T \leqslant U$ if and only if $T \ll_M U$.*

Proof. ($\Rightarrow$): Suppose $T \leqslant U$. We use induction on s to show that $T \downarrow s$ implies both $U \downarrow s$ and $\mathscr{A}(U, s) \subseteq \mathscr{A}(T, s)$. If $T \downarrow \varepsilon$ this means that $unfold(T)$ has the form $(\mathscr{A}, f)$. So $unfold(U)$ must have the form $(\mathscr{B}, g)$, where $\mathscr{B} \subseteq \mathscr{A}$ and $f \leqslant g$. This in turn means that $U \downarrow$ and $\mathscr{A}(U, \varepsilon) \subseteq \mathscr{A}(T, \varepsilon)$. Suppose $T \downarrow as$. Then $T \downarrow$ and therefore $U \downarrow$. Now suppose $U \xrightarrow{a} U'$. Then $T \xrightarrow{a} T'$ for some T' such that $T' \leqslant U'$. We know that $T' \downarrow s$ and by induction on s, this is sufficient to ensure that $U' \downarrow s$, i.e., $U \downarrow as$ and $\mathscr{A}(U', s) \subseteq \mathscr{A}(T', s)$, i.e., $\mathscr{A}(U, as) \subseteq \mathscr{A}(T(as))$.

($\Leftarrow$): In this direction we need to appeal to some general results about initial solutions in <u>CPO</u>. For any T in AT_s^v let

(i) $T^0 = \perp$

(ii) (a) if $unfold(T) = \perp$ then $T^{n+1} = \perp$

(b) otherwise $unfold(T) = (\mathscr{A}, f)$ and $T^{n+1} = fold((\mathscr{A}, f_n))$ where f_n is defined pointwise by

$$f_n(c!) = \{\langle v, (f(c!)(v))^n \rangle \mid v \in domain(f(c!))\}$$

$$f_n(c?) = \lambda v.(f(c?)(v))^n.$$

Then, because AT_s^v is the limit of the functor F, if for every $n \geqslant 0$ $T^n \leqslant U^n$ it follows that $T \leqslant U$. However it is straightforward to show, by induction on n, that $T \ll_M U$ implies $T^n \leqslant U^n$ for every $n \geqslant 0$. ∎

We now have a relation $\ll_M$ defined on processes and on trees; the former characterizes the behavioural preorder $\sqsubseteq_M$ while the latter characterizes the semantic ordering. In Section 4 we prove that for all processes p, q, $p \ll_M q$ if and only if $AT_s^v[\![p]\!] \ll_M AT_s^v[\![q]\!]$ where $AT_s^v[\![p]\!]$ is an interpretation of p in AT_s^v, thereby proving full-abstraction of the model with respect to testing. In the next section we explain how to interpret processes in the model.

3.3. *Strong Acceptance Trees as a Natural Interpretation*

The language VPL can be interpreted in any natural interpretation and so in order to use AT_s^v to model VPL it is sufficient to show how it can be viewed as a natural interpretation. This amounts to giving a continuous function, $op_{AT_s^v}$, for each operator symbol in Σ and in addition input and output functions of the correct type. To ease the notation difficulties in this subsection we abbreviate AT_s^v to AT. To make the definitions more compact we introduce the following convention: if h is a function from

$AT^n \to AT$ we also use h to denote the natural extension $ext(h)$: $(Val \to AT)^n \to (Val \to AT)$ defined by

$$ext(h)(f_1, ..., f_n)(v) = h(f_1(v), ..., f_n(v)).$$

We also use $cl(X)$ to denote the least saturated set containing the finite collection of finite sets X. We refrain from offering motivation for the definitions to follow. This may be found in [He88], although we borrow much of the notation from [Miln88].

Output.

Define $out_{AT} : Chan \times Val \times AT \to AT$ by

$$out_{AT}(c, v, t) = \langle \{\{c!\}\}, f \rangle$$

where

$$domain(f) = \{c!\}$$
$$f(c!) = \{\langle v, t \rangle\}.$$

Input.

Define $in_{AT} : Chan \times (Val \to AT) \to AT$ by

$$in_{AT}(c, g) = \langle \{\{c?\}\}, f \rangle$$

where

$$domain(f) = \{c?\}$$
$$f(c?) = g.$$

NIL.

Let NIL_{AT} be the tree $\langle \{\{\ \}\}, f \rangle$, where f is the empty function.

Ω.

Let Ω_{AT} be $\perp_{AT}$.

Internal Nondeterminism.

Define $\oplus_{AT} : AT \times AT \to AT$ as

$$Y\lambda I.\lambda t.\lambda u. \quad \begin{array}{l} \textit{if } t = \perp \textit{ or } u = \perp \\ \quad \textit{then } \perp \\ \quad \textit{else let} \\ \qquad \langle \mathscr{A}, f \rangle = t \\ \qquad \langle \mathscr{B}, g \rangle = u \\ \qquad \textit{in } \langle c(\mathscr{A} \cup \mathscr{B}), h \rangle \end{array}$$

where h is defined by

$$h(c?) = I(f(c?), g(c?)) \text{ if } c? \in domain(f) \cap domain(g)$$
$$= f(c?) \text{ if } c? \in domain(f) \setminus domain(g)$$
$$= g(c?) \text{ if } c? \in domain(g) \setminus domain(f)$$

and

$$h(c!) = f(c!) \text{ if } c! \in domain(f) \setminus domain(g)$$
$$= g(c!) \text{ if } c! \in domain(g) \setminus domain(f)$$
$$= k \text{ if } c! \in domain(f) \cap domain(g)$$

where

$$k(v) = I(f(c!)(v),\ g(c!)(v))$$
$$\text{if } v \in domain(f(c!)) \cap domain(g(c!))$$
$$= f(c!)(v) \text{ if } v \in domain(f(c!)) \setminus domain(g(c!))$$
$$= g(c!)(v) \text{ if } v \in domain(g(c!)) \setminus domain(f(c!)).$$

External Nondeterminism.

Define $+_{AT} : AT \times AT \to AT$ by

$$t +_{AT} u = \textit{if } t = \bot \textit{ or } u = \bot$$
$$\textit{then } \bot$$
$$\textit{else let}$$
$$\langle \mathscr{A}, f \rangle = t$$
$$\langle \mathscr{B}, g \rangle = u$$
$$\textit{in } \langle \mathscr{A} \cup \mathscr{B}, h \rangle$$

where

$$\mathscr{A} \cup \mathscr{B} = \{ A \cup B \mid A \in \mathscr{A},\ B \in \mathscr{B} \}$$

and h is defined by

$$h(a) = f(a) \text{ if } a \in domain(f) \setminus domain(g)$$
$$= g(a) \text{ if } a \in domain(g) \setminus domain(f)$$
$$h(c?) = f(c?) \oplus_{AT} g(c?) \text{ if } c? \in domain(f) \cap domain(g)$$

and for $c! \in domain(f) \cap domain(g)$, $h(c!)$ is defined by

$$h(c!)(v) = f(c!)(v) \oplus_{AT} g(c!)(v)$$
$$\text{if } v \in domain(f(c!)) \cap domain(g(c!))$$
$$= f(c!)(v) \text{ if } v \in domain(f(c!)) \setminus domain(g(c!))$$
$$= g(c!)(v) \text{ if } v \in domain(g(c!)) \setminus domain(f(c!)).$$

Restriction.

For each $c \in Chan$ let $\setminus_{AT} c : AT \to AT$ denote

$$Y\lambda R.\lambda t.\quad \textit{if } t = \bot \textit{ then } \bot$$
$$\textit{else let } \langle \mathscr{A}, f \rangle = t$$
$$\textit{in } \langle \mathscr{B}, g \rangle$$

where

$$\mathscr{B} = \{ A \setminus \{c?, c!\} \mid A \in \mathscr{A} \}$$

and

$$g(e) = R(f(e)) \text{ for } e \in Ev(\mathscr{B}).$$

Renaming.

For any finite permutation, R, of channel names, let $[R]_{AT} : AT \to AT$ denote

$$Y\lambda F.\lambda t.\quad \textit{if } t = \bot \textit{ then } \bot$$
$$\textit{else let } \langle \mathscr{A}, f \rangle = unfold(t)$$
$$\textit{in } fold(\langle \mathscr{B}, g \rangle)$$

where

$$\mathscr{B} = \{ \{ R(e) \mid e \in A \} A \in \mathscr{A} \}$$

and

$$g(e) = F(f(R^{-1}(e))) \text{ for } e \in Ev(\mathscr{B}).$$

Parallel.

Define $|_{AT} : AT \times AT \to AT$ as

$$Y\lambda F.\lambda t.\lambda u. \quad \textit{if } t = \bot \textit{ or } u = \bot \textit{ then } \bot \textit{ else let } \langle \mathscr{A}, f \rangle = t \; \langle \mathscr{B}, g \rangle = u \textit{ in } \sum \{ t_{AB} \mid A \in \mathscr{A}, B \in \mathscr{B} \}$$

where

$$t_{AB} = \textit{if } INT(A, B) = \varnothing \textit{ then } sumext(A, B) \textit{ else } (sumext(A, B) + sumint(A, B)) \oplus sumint(A, B)$$

where

$$sumext(A, B) = \sum EXT(A, B)$$
$$sumint(A, B) = \sum INT(A, B)$$

where $INT(A, B)$, $EXT(A, B)$ are defined by

$$INT(A, B) = \{ F(f(c?)(v), g(c!)(v)) \mid c? \in A, c! \in B \textit{ and } v \in domain(g(c!)) \} \cup \{ F(f(c!)(v), g(c?)(v)) \mid c! \in A, c? \in B \textit{ and } v \in domain(f(c!)) \}$$

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

These definitions are notationally quite complex but the reformulations of Acceptance Trees enable us to be much more precise than in [He88].

PROPOSITION 3.5. *AT_s^v is a Natural Interpretation.*

Proof. It is necessary to show that each of the functions defined above is continuous. This is rather tedious and is omitted. ∎

To end this section we show that each finite element in the ω-algebraic *cpo* AT_s^v is definable in our language. Let NDL denote the sublanguage of VPL obtained by omitting the operators $\backslash c$, $[R]$, $|$, $recP$. .

THEOREM 3.6. *For each finite element d in AT_s^v there is a term t in NDL such that*

$$AT_s^v[\![t]\!] = D.$$

Proof. In $\underline{\underline{CPO}}$ AT_s^v is the limit of the sequence of domains AT^n, $n \geqslant 0$, where

$$AT^0 = \{\perp\},$$
$$AT^{n+1} = F(AT^n),$$

and the finite elements of AT_s^v are simply the union of the finite elements of each AT^n. The unique finite element of AT^0 is obviously definable. So assume that the finite elements of AT^k are definable. We show that those of AT^{k+1} are also definable.

Note that any element of AT^{k+1} can be characterized by an acceptance set $\mathscr{A}$ and a list containing a unique entry for each $a \in Ev(\mathscr{A})$, of the form

$$\langle c_1?, f_1\rangle, \ldots, \langle c_l?, f_l\rangle, \langle c_1'!, g_1\rangle, \ldots, \langle c_m'!, g_m\rangle,$$

where each f_i is finite element in $Val \to AT^k$ and each g_j is an element of $Fin(Val - AT^k)$ whose range is contained in the subset of finite elements of AT^k. First consider a typical f_i. It sends almost all values to $\perp$ and therefore can be characterized by a finite nonrepeating list $(v_1, f_i(v_1)), \ldots, (v_{m_i}, f_i(v_{m_i}))$, where $\{v_1, \ldots, v_{m_i}\} = \{v \in Val \mid f_i(v) \neq \perp\}$. Each $f_i(v_j)$ is a finite element in AT^k and so by induction we may assume that it is denoted by a term q_j. Then let $t(c_i?)$ denote the term

$$\begin{array}{ll} c_i?\, x. & \textit{If } x = v_1 \textit{ Then } q_1 \\ & \textit{Else If } x = v_2 \textit{ Then } \ldots \\ & \vdots \\ & \textit{Else } \Omega. \end{array}$$

Now let us consider a typical g_j. Once more this can be characterized by a finite nonrepeating list

$$(v_1, g_j(v_1)), \ldots, (v_{l_j}, g_j(v_{l_j})),$$

where $domain(g) = \{v_1, \ldots, v_{l_j}\}$. Once more each $g_j(v_i))$ is a finite element in AT^n and by induction we may assume it is denoted by a term r_i. In this case let $t(c_j!)$ be the term

$$c_j!\, v_1.r_1 + \cdots + c_j!\, v_{l_j}.r_{l_j}.$$

Then the finite element of AT^{k+1} described above is denoted by the term

$$\sum \left\{ \sum \{t(e) \mid e \in A\} \mid A \in \mathscr{A} \right\},$$

where $\sum$ is the usual symbol for finite summation with respect to $+$ and $\sum$ w.r.t. $\oplus$. ∎

4. The Proof System

In this section we give a sound and complete proof system for processes with respect to the model AT_s^v. It will follow from the full abstraction result of Section 4.3 that it is also sound and complete with respect to the behavioural ordering $\sqsubseteq_M$. The proof system is derived from that in Chapter 5 of [He88], with which it has much in common, including the infinitary rule to deal with recursive terms. The only significant addition is the rule

$$\frac{t[v/x]\, s \sqsubseteq u[v/x], \text{ for every } v \in Val}{c?x.t\, s \sqsubseteq c?x.u}$$

Of course this is also an infinitary rule if Val is infinite. In a more realistic proof system it would be replaced by a finitary form of induction appropriate to the data type of Val. In addition to this rule we have the extended equations discussed in the introduction, the obvious modification to the interleaving law of [He88], and a number of rules concerned with the evaluation of expressions and boolean expression.

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 $op \in \Sigma$	
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	$\dfrac{t \sqsubseteq u}{t\rho \sqsubseteq u\rho}$	$\dfrac{}{t \sqsubseteq u}$ for every equation $t \sqsubseteq u$
VI	$\dfrac{d \sqsubseteq u \text{ for every } d \in App(t)}{t \sqsubseteq u}$	
VII	$\dfrac{[\![e]\!] = [\![e']\!]}{c!e.t = c!e'.t}$	
VIII	$\dfrac{[\![be]\!] = T}{If\ be\ Then\ t\ Else\ u = t}$	$\dfrac{[\![be]\!] = F}{If\ be\ Then\ t\ Else\ u = u}$
IX	$\dfrac{}{c?x.t = c?y.t[y/x]}$ if y does not occur free in t	

FIG. 4. Proof system.

$$
\begin{aligned}
X \oplus (Y \oplus Z) &= (X \oplus Y) \oplus Z \\
X \oplus Y &= Y \oplus X \\
X \oplus X &= X \\
X + (Y + Z) &= (X + Y) + Z \\
X + Y &= Y + X \\
X + X &= X \\
X + NIL &= X \\
pre.X + pre.Y &= pre.(X \oplus Y) \\
c?x.X + c?x.Y &= c?x.X \oplus c?x.Y \\
c!e.X + c!e'.Y &= c!e.X \oplus c!e'.Y \\
X + (Y \oplus Z) &= (X + Y) \oplus (X + Z) \\
X \oplus (Y + Z) &= (X \oplus Y) + (X \oplus Z) \\
X \oplus Y &\sqsubseteq X \\
X + \Omega &\sqsubseteq \Omega \\
\Omega &\sqsubseteq X
\end{aligned}
$$

FIG. 5. Basic equations.

In Section 4.1 we describe the proof system and prove it sound w.r.t. the model AT_s^r. Completeness is the topic in Section 4.2. In the final subsection, Section 4.3, we use the proof system to show that the model is also fully abstract w.r.t. $\lesssim_M$, thereby uniting the three different view of processes. Throughout we abbreviate $AT_s^r[\![t]\!]$ to $[\![t]\!]$.

4.1. *Definition of the Proof System*

The proof system is equationally based and is given in Fig. 4. The rules should be self-explanatory. Note that Rules VII, VIII assume knowledge of the evaluation of expressions, and that Rule IX is an α-conversion rule for input terms. The equations are given in Figs. 5–7. Figure 5 is essentially

$$
\begin{aligned}
(X \oplus Y)\backslash c &= X\backslash c \oplus Y\backslash c \\
(X \oplus Y)[R] &= X[R] \oplus Y[R] \\
(X + Y)\backslash c &= X\backslash c + Y\backslash c \\
(X + Y)[R] &= X[R] + Y[R] \\
(pre.X)\backslash c &= \begin{cases} pre.(X\backslash c) & \text{if } c \neq chan(pre) \\ NIL & \text{otherwise} \end{cases} \\
(pre.X)[R] &= R(pre).(x[R]) \\
NIL\backslash c &= NIL \\
NIL[R] &= NIL \\
\Omega\backslash c &= \Omega \\
\Omega[R] &= \Omega \\
(X \oplus Y) \mid Z &= X \mid Z \oplus Y \mid Z \\
X \mid (Y \oplus Z) &= X \mid Y \oplus X \mid Z \\
NIL \mid P &= P \mid NIL = P \\
X \mid (Y + \Omega) &= (X + \Omega) \mid Y = \Omega
\end{aligned}
$$

FIG. 6. More equations.

Let X, Y denote $\sum\{pre_i.X_i, i\in I\}$, $\sum\{pre_j.Y_j, j\in J\}$. Then

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

where

$$ext(X, Y) = \sum\{pre_i.(X_i \mid Y), i\in I\} + \sum\{pre_j.(X \mid Y_j), j\in J\}$$

$$int(X, Y) = \sum\{X_i[v/x] \mid Y_j, pre_i = c?x, pre_j = c!v\}$$

$$\oplus \sum\{X_i \mid Y_j[v/y], pre_i = c!v, pre_j = c?y\}$$

FIG. 7. Interleaving law.

concerned with the interplay between the nondeterministic operators and prefixing while in Figure 6 we give the equations governing restriction, renaming, and parallel. Figure 7 gives an obvious adaptation of the interleaving law in Fig. 5.3 in [He88] to handle value-passing. In Fig. 6 we use $chan(pre)$ to denote the channel occurring in the prefix pre and $R(pre)$ to denote the obvious extension of the channel renaming R to prefixes. In Fig. 7 the predicate *comms* is defined to be true only if for some i, j pre_i and pre_j are inverses of each other.

Let $\vdash t \sqsubseteq u$ denote the fact that $t \sqsubseteq u$ can be derived from the proof system.

THEOREM 4.1. $\vdash t \sqsubseteq u$ *implies* $[\![t]\!] \leqslant [\![u]\!]$.

Proof. First it is necessary to check that AT_s^v satisfies all of the equations, which is redious but not very difficult apart from the interleaving law. Then it is necessary to check that all of the rules are sound in AT_s^v. All are straightforward. For Example, VI is sound because in any natural interpretation D

$$D[\![t]\!] = \bigsqcup\{D[\![d]\!], d\in App(t)\}. \quad \blacksquare$$

4.2. *Completeness of the Proof System*

In this section we show the completeness of the proof system with respect to the model AT_s^v; i.e.,

$$[\![p]\!] \leqslant [\![q]\!] \text{ implies } \vdash p \sqsubseteq q.$$

It is sufficient to show a slightly weaker result: for every finite term d and process q,

$$[\![d]\!] \leqslant [\![q]\!] \text{ implies } \vdash d \sqsubseteq q. \tag{3}$$

For suppose we have proved (3) and $[\![p]\!] \leqslant [\![q]\!]$. Now

$$[\![p]\!] = \bigsqcup \{[\![p^n]\!], n \geqslant 0\}.$$

So, for every $d \in App(p)$, $[\![d]\!] \leqslant [\![q]\!]$. By (3) it follows that for every $d \in App(p)$, $\vdash d \sqsubseteq q$. Applying Rule VI we obtain the required $\vdash p \sqsubseteq q$.

To prove partial completeness (3) we do not require the full proof system; more specially, we do not require Rule VI. Let $p \sqsubseteq_A q$ denote the fact that $p \sqsubseteq q$ can be derived in the proof system without the use of Rule VI or the first part of IV.

We show that

$$[\![d]\!] \leqslant [\![q]\!] \text{ implies } d \sqsubseteq_A q.$$

The central notion is that of *head normal form.*

DEFINITION 4.2. (1) *NIL* is h.n.f..

(2) Let $\mathscr{A}$ be a saturated set, and for each $e \in Ev(\mathscr{A})$, let t_e be a term given as follows:

(a) If $e = c?$ then t_e is $c?x.t'$ for some term t'.

(b) If $e = c!$ then t_e is $\sum \{c!v.f(v) \mid v \in domain(f)\}$, where f is a partial function from *Val* into VPL with a finite domain; i.e., $f \in Fin(Val \rightharpoonup \text{VPL})$.

Then

$$\sum \left\{ \sum \{t_e \mid e \in A\} \mid A \in \mathscr{A} \right\}$$

is a h.n.f.

We refer to t' above as $t(c?)$ and f as $t(c!)$.

Note the similarity of the h.n.f.'s to the trees in AT_s^v. We wish to show that every convergent term has a h.n.f. and the proof follows closely the corresponding result for the pure language. There the proof used three derived equations:

$$X \oplus Y = X \oplus Y \oplus (X + Y) \qquad \text{Der 1}$$

$$\begin{aligned} & X \oplus (X + Y + Z) \\ = {} & X \oplus (X + Y) \oplus (X + Y + Z) \end{aligned} \qquad \text{Der 2}$$

$$\begin{aligned} & (pre.X_1 + Z_1) \oplus (pre.X_2 + Z_2) \\ = {} & (pre.X + Z_1) \oplus (pre.X + Z_2) \\ & \text{where } X \text{ is } X_1 \oplus X_2 \end{aligned} \qquad \text{Der 3.}$$

These are also derived equations in our proof system but, in addition, we also require a modification of Der 3 for output guards:

$$(c!e.X_1 + Z_1) \oplus (c!e'.X_2 + Z_2) = (W + Z_1) \oplus (W + Z_2)$$
$$\text{where } W \text{ is } c!e.X_1 + c!e'.X_2 \qquad \text{Der } 3'.$$

This may be derived in a similar fashion to Der 3 (see p. 97 of [He88]):

$$\begin{aligned}
&(c!e.X_1 + Z_1) \oplus (c!e'.X_2 + Z_2) \\
&= (c!e.X_1 + Z_1) \oplus (c!e'.X_2 + Z_2) \\
&\quad \oplus (W + Z_1 + Z_2) && \text{by Der 1} \\
&= (c!e.X_1 + Z_1) \oplus (W + Z_1) \oplus (c!e'.X_2 + Z_2) \\
&\quad \oplus (W + Z_2) \oplus (W + Z_1 + Z_2) && \text{by Der 2} \\
&= (c!e.X_1 + Z_1) \oplus (W + Z_1) \oplus (c!e'.X_2 + Z_2) \\
&\quad \oplus (W + Z_2) && \text{by Der 1} \\
&= ((c!e.X_1 \oplus W) + Z_1) \oplus ((c!e'.X_2 \oplus W) + Z_2) \\
&\qquad \text{using the distributivity of } + \text{ over } \oplus.
\end{aligned}$$

But with the new equation, $c!e.X + c!e'.Y = c!e.X \oplus c!e'.Y$, one can show that

$$c!e.X_1 \oplus W = c!e'.X_2 \oplus W = W$$

and the result follows.

PROPOSITION 4.3. *If* $p\downarrow$, *then there exists a h.n.f.,* $hnf(p)$, *such that* $p =_A hnf(p)$.

Proof (Outline). The proof proceeds in two stages. First processes are reduced to *sum forms* and then the latter are converted to h.n.f.'s. A sum form is a process whose toplevel operators are prefixing, $+$, or $\oplus$. Formally it is the least set of processes SF which satisfies:

1. $NIL \in SF$, $pre.t \in SF$.
2. $p, q \in SF$ implies $p + q \in SF$ and $p \oplus q \in SF$.

The technique of Theorem 2.4.6 of [He88] may be used to convert a sum form to a h.n.f. but using Der 3′ in place of Der 3. So we concentrate on showing how to convert processes into sum forms.

Let $<$ be the least transitive relation which satisfies

1. $p < op(\cdots, p, \cdots)$ if $op \in \{|, +, \oplus, \backslash c, [R]\}$
2. $p \rightarrowtail q$ implies $p < q$.

We use $Ind(<)$ to denote the set of processes p such that $\{p' \mid p' < p\}$ is finite. It is easy to show, by structural induction, that if $p \notin Ind(<)$ then $p \rightarrowtail q$ for some q such that $q \notin Ind(<)$. In other words $p\downarrow$ implies $p \in Ind(<)$ and therefore we may prove the required statement using induction over $<$.

The proof proceeds by a case analysis on the dominant operator in p and in the cases of the operators $|$, $[R]$, and $\backslash c$ they can be eliminated using the equations of Figs. 6 and 7. The only difficult case is the parallel combinator $|$ for which we must phrase the inductive hypotheses in such a way that, when we apply the interleaving law, further occurrences of $|$ may be eliminated. The required inductive hypothesis is: For each $p \in Ind(<)$ there is a sum form, $sf(p)$, such that

1. $p =_A sf(p)$
2. for every $s \in Act^+$, $sf(p) \stackrel{s}{\Longrightarrow} r$ implies $p \stackrel{s}{\Longrightarrow} r$. ∎

We also need the following property of finite terms.

LEMMA 4.4. *For every closed finite term d*

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

Proof. By repeated use of the interleaving law we can eliminate the operator $|$ from d. The proof then proceeds by structural induction on d. For each of the operators there is an equation which states that it is strict in all of its arguments. ∎

A similar relationship between behavioural convergence and denoting $\perp$ in AT_δ^τ can be established for arbitrary closed terms.

LEMMA 4.5. *For every closed term p $p\downarrow$ if and only if $[\![p]\!]\downarrow$.*

Proof. First suppose $p\downarrow$. Then p has a h.n.f. $hnf(p)$ and, since the proof system is sound with respect to AT_δ^τ, $[\![p]\!] = [\![hnf(p)]\!]$. But because of the structure of h.n.f.'s it is simple to show that they are intrepreted as trees T such that $T\downarrow$. Therefore $[\![p]\!]\downarrow$.

To prove the converse first consider finite closed terms d. If $d\uparrow$ then $d =_A \Omega$ and since $[\![\Omega]\!] = \perp$ it follows that $[\![d]\!] = \perp$. So $[\![d]\!]\downarrow$ implies $d\downarrow$. Now suppose that for some closed term p, $[\![p]\!]\downarrow$. Then since $[\![p]\!] = \bigsqcup\{[\![p^n]\!] \mid n \geqslant 0\}$, there exists some $n > 0$ such that $[\![p^n]\!]\downarrow$. Since p^n is finite this implies $p^n\downarrow$. Now $p^n \sqsubseteq_M p$ and this in turn implies that $p^n \ll_M p$. So we can conclude that $p\downarrow$. ∎

THEOREM 4.6. *For any closed finite term d and closed term q,*

$$[\![d]\!] \leqslant [\![q]\!] \text{ implies } d \sqsubseteq_A q.$$

Proof. By induction on the depth of d, i.e., the length of the longest sequence of external moves it can make. Note that d and $hnf(d)$, if it exists, have the same depth.

If $d\uparrow$ then since $d =_A \Omega$ the result follows trivially. So we may assume that $d\downarrow$. From Lemma 4.5 this in turn implies that $q\downarrow$. So let their respective h.n.f.'s be

$$\sum\left\{\sum\{d_e \mid e \in A\} \mid A \in \mathscr{A}\right\}$$

$$\sum\left\{\sum\{q_e \mid e \in B\} \mid B \in \mathscr{B}\right\}.$$

Because of the structure of h.n.f.'s

$$\mathscr{A}([\![d]\!], \varepsilon) = \mathscr{A},$$

$$[\![d]\!] \xrightarrow{c!v} [\![d(c!)(v)]\!] \quad \text{for} \quad c! \in Ev(\mathscr{A}) \text{ and } v \in domain(d(c!))$$

$$[\![d]\!] \xrightarrow{c?v} [\![d(c?)[v/x]]\!] \text{ for } c? \in Ev(\mathscr{A}),$$

where x is the variable $c?$ binds in $d(c?)$ and similarly for q. By the definition of $\leqslant$ in AT_v^τ it follows that $\mathscr{B} \subseteq \mathscr{A}$, and for each $c! \in Ev(\mathscr{B})$ and $v \in domain(q(c!))$ $[\![d(c!)(v)]\!] \leqslant [\![q(c!)(v)]\!]$ and for each $c? \in Ev(\mathscr{B})$ $[\![d(c?)[v/x]]\!] \leqslant [\![q(c?)[v/x]]\!]$ for every $v \in Val$. By induction we have for such $c!$, v and $c?$ that $d(c!)(v) \sqsubseteq_A q(c!)(v)$ and $d(c?)[v/x] \sqsubseteq_A q(c?)[v/x]$. Using Rule III we can now show that for each $c?$ in $Ev(B)$ $d_{c?} \sqsubseteq_A q_{c?}$. Similarly we can use the equations, in particular $X \oplus Y \leqslant X$ and the new equation $c!e.X \oplus c!e'.Y = c!e.X + c!e'.Y$, to show $d_{c!} \sqsubseteq_A q_{c!}$. We may now proceed as in [He88] to show $d \sqsubseteq_A q$:

$$d =_A \sum\left\{\sum\{d_e \mid e \in A\} A \in \mathscr{A}\right\}$$

$$=_A \sum\left\{\sum\{d_e \mid e \in A\} A \in \mathscr{B}\right\} \oplus r \qquad \text{for some } r$$

$$\sqsubseteq_A \sum\left\{\sum\{d_e \mid e \in A\} A \in \mathscr{B}\right\}$$

$$\sqsubseteq_A \sum\left\{\sum\{q_e \mid e \in A\} A \in \mathscr{B}\right\}$$

$$=_A q. \quad \blacksquare$$

Following the discussion at the beginning of this subsection we can now state:

THEOREM 4.7. *For all processes p, q we have that $\vdash p \sqsubseteq q$ if and only if $[\![p]\!] \leqslant [\![q]\!]$.*

4.3. *Full Abstraction*

In this section we show that the model AT_s^v is fully abstract with respect to the behavioural preorder $\sqsubseteq_M$; i.e.,

$$p \sqsubseteq_M q \text{ if and only if } [\![p]\!] \leqslant [\![q]\!]. \tag{4}$$

This in turn implies the soundness and the completeness of the proof system with respect to $\sqsubseteq_M$, because of Theorem 4.7.

The key to proving this result is the relation $\ll_M$. It is defined both on terms and on trees and in each case is defined in terms of convergence predciates $\downarrow s$, for each sequence of actions s, and acceptance sets. We have already shown that

$$p \sqsubseteq_M q \Leftrightarrow p \ll_M q$$

for all closed terms p, q, and

$$T \leqslant U \Leftrightarrow T \ll_M U$$

for all trees T, U. So we show (4) by proving

$$p \ll_M q \Leftrightarrow [\![p]\!] \ll_M [\![q]\!].$$

This in turn follows from

$$p \downarrow s \Leftrightarrow [\![p]\!] \downarrow s$$

and

$$\mathscr{A}([\![p]\!](s)) = cl(\mathscr{A}(p, s)),$$

where cl is the closure operator for saturated sets referred to in Section 3.3. Recall that $cl(X)$ is the least collection of finite subsets such that

1. $X \in cl(X)$.
2. $A, C \in cl(X)$ and $A \subseteq B \subseteq C$ implies $B \in cl(X)$.
3. $A, B \in cl(X)$ implies $A \cup B \in cl(X)$.

These two results are not shown directly for all processes p; they are mediated by the useful notion of head-normal forms. We leave the reader to check that $p \sqsubseteq_A q$ implies $p \sqsubseteq_M q$; this means checking that $\sqsubseteq_M$ is satisfied for all the equations and is preserved by the appropriate rules. It then follows that $p \simeq_M hnf(p)$.

PROPOSITION 4.8. *For every closed term p,*

$$p \downarrow s \Leftrightarrow [\![p]\!] \downarrow s.$$

Proof. The proof is by induction on the length of s. The base case, when $s = \varepsilon$, has already been shown in Lemma 4.5. So we assume that s has the form as'.

(i) Suppose $p \downarrow as'$. Then $p \downarrow$ and so has h.n.f. $hnf(p)$. Now by the construction of the h.n.f.'s:

$$hnf(p) \xrightarrow{a} q \Leftrightarrow [\![p]\!] \xrightarrow{a} [\![q]\!].$$

So if $[\![hnf(p)]\!] \xrightarrow{a} T$ then $hnf(p) \xrightarrow{a} q$ such that $[\![q]\!] = T$. Now $q \downarrow s'$ and so by induction $T \downarrow s'$. This means $[\![hnf(p)]\!] \downarrow as'$ and therefore $[\![p]\!] \downarrow as'$.

(ii) Suppose $[\![p]\!] \downarrow as'$. The proof that $p \downarrow as$ is very similar. ∎

PROPOSITION 4.9. *For every $s \in Act^*$, $p \downarrow s$ implies that $\mathscr{A}([\![p]\!], s) = cl(\mathscr{A}(p, s))$.*

Proof. Again by induction on the length of s.

(i) $s = \varepsilon$. Since $p \downarrow$ it has a h.n.f. $hnf(p)$ and $[\![p]\!] = [\![hnf(p)]\!]$. By the construction of h.n.f.'s $\mathscr{A}([\![hnf(p)]\!], \varepsilon) = \mathscr{A}(hnf(p), \varepsilon)$. Also since $p =_A hnf(p)$ it follows that $p \ll_M hnf(p)$ and $hnf(p) \ll_M p$. This in turn means that $cl(\mathscr{A}(p, \varepsilon)) = cl(\mathscr{A}(hnf(p), \varepsilon))$. But by the definition of h.n.f., $cl(\mathscr{A}(hnf(p), \varepsilon) = \mathscr{A}(hnf(p), \varepsilon)$, from which it follows that $\mathscr{A}([\![p]\!], \varepsilon) = cl(\mathscr{A}(p, \varepsilon))$.

(ii) $s = as'$. Again $p \downarrow s$ means $p \downarrow$ and therefore it has a h.n.f. $hnf(p)$ with $[\![p]\!] = [\![hnf(p)]\!]$. Here we use again the property

$$hnf(p) \xrightarrow{a} q \Leftrightarrow [\![hnf(p)]\!] \xrightarrow{a} [\![q]\!].$$

So by induction, if either of $\mathscr{A}(p, as)$ or $\mathscr{A}([\![p]\!], as)$ is nonempty, then both are nonempty. So let us assume the latter. Then

$$\begin{aligned} cl(\mathscr{A}(p, as')) &= cl(\mathscr{A}(q, s')) && \text{where } hnf(p) \xrightarrow{a} q \\ &= \mathscr{A}([\![q]\!], s') && \text{by induction} \\ &= \mathscr{A}([\![hnf(p)]\!], as') \\ &= \mathscr{A}([\![p]\!], as'). \end{aligned}$$ ∎

Again from the discussion at the beginning of the subsection we can conclude that

THEOREM 4.10. *For all processes p, q,*

$$p \sqsubseteq_M \text{ if and only if } [\![p]\!] \leqslant [\![q]\!].$$

5. Conclusion

We have presented a semantic theory of a process description language which supports value-passing. It consists of a denotational model which is fully abstract with respect to a natural notion of testing and a proof system which is both sound and complete with respect to the model. One can use Theorem 3.6 to characterize the model further: it is the initial Natural Interpretation which is fully abstract with respect to testing.

Although the language is theoretically quite powerful, it is very limited from a practical point of view, as the examples in Section 2.1 show. A more natural language would, for example, allow process names to be parameterized by values and would have values of different types, with specific types associated with individual channels. These additions would improve the language considerably and all could easily be accommodated within our theory. It should also be apparent that the theory does not depend in any way on the use of the particular combinator $|$. Many other parallel combinators from the literature may also be modeled for the reasons explained in [He88]: essentially we have a theory of nondeterministic processes on top of which one may model a wide variety of parallel combinators.

The complete proof system is mainly of theoretical interest, although, if the two infinitary rules are replaced by finitary forms of induction, the result would be powerful and somewhat more useful. It would still presuppose the ability to evaluate expressions. A more interesting proof system would incorporate a subproof system for the equality of value-expressions which may include free variables and would prove statements of the form

$$e_1 = e'_1, \ldots, e_k = e'_k \vdash p = q.$$

Such a proof system is defined in [He89].

The research reported here was motivated by the publication of [Miln88] where a minor variation on the model AT^V_ω was originally presented. We have offered a formal justification for this model, namely, that it identifies processes which cannot be differentiated using a natural notion of testing. As far as we are aware, no similar result appears in the literature for process languages which support value-passing.

There are, however, various semantic theories of such languages. In [Mil80] an operational semantics similar to ours is given for "full CCS" and in [HP80] an observational preorder for this language is characterized equationally. Indeed, this result is very similar to the completeness theorem for our proof system. However, this theory lacks a natural model, although a term model is constructed from the equational characterization. More

recent work on "full CCS" (see [Mil89]) tends to explain value-passing in terms of infinite choice, with $c?x.p$ being modeled by

$$\left\{\sum c_v.\, p[v/x] \mid v \in Val\right\}.$$

This approach, which is based on the theory of bisimulations, also lacks natural models and complete algebraic characterizations are difficult to obtain. However, it is natural to wonder to what extent this derived notion of value-passing could be accommodated within the standard framework of testing [He88]. Both the full-abstractness and completeness results in this framework depend crucially on the fact that processes are finite-branching. To be precise, in order to interpret $c?x.p$ in the above fashion the language needs to have an infinite summation operator, $\sum_{v \in Val}$, at least if Val is infinite. Such an operator cannot be interpreted in the standard model AT_s because each tree in AT_s has a finite number of successor branches.

One could still use this derived notion of value-passing in the framework of testing if one could find a simple generalization of AT_s which dropped the assumption of finite branching. This would avoid the necessity of defining the more subtle variant AT_s^v, particularly if the extension were straightforward. However, there are at least two problems with this suggestion. The first is that if such a generalization could be found then it would not be fully abstract with respect to the behavioural preorder $\sqsubseteq_M$ for the simple reason that it would not satisfy the axiom

$$c!e.X + c!e'.Y = c!e.X \oplus c!e'.Y.$$

This axiom is a consequence of our natural framework of testing, in which instances of the left and right hand sides cannot be differentiated. It is conceivable that one might be interested in a more powerful notion of testing where, for example, the processes $c!1.p + c!2.q$ and $c!1.p \oplus c!2.q$ could be distinguished. This essentially means that one has processes with unbounded nondeterminism and moreover observers can discern this unbounded behaviour. This brings us to the second problem, namely that an infinitary version of AT_s is not easy to define. One suggestion would be to take the definition of AT_s but omit the requirements about finiteness. However, this would mean that the order defined in Section 3 would not be complete. One can see this without being too precise. Let T_k be the denotation in the proposed model of the "term"

$$\sum \{a_n.NIL \mid n \geqslant k\}.$$

Then $T_0 \leqslant T_1 \leqslant \cdots \leqslant T_k \leqslant \cdots$ forms a chain which has no limit. One could try to invent a new order and this is the approach taken in [Ros88b]. Here the standard ordering on AT_s, or more accurately, the failures model of CSP, which is very similar, is divided into its two components, one of which is the "improvements order," which models the fact that one process is more deterministic than another; the other, called the "definedness order," is concerned only with divergence. A similar approach could be taken to the infinitary version of AT_s, where the corresponding version of the "definedness" order would be complete. However, there are still problems. Using this derived notion of value-passing the hiding or restricting of channels must be modeled by hiding or restricting with respect to an infinite number of actions, which is not continuous even with respect to the "definedness" order. For example, let P_n be the denotation of the term

$$\sum \{c_k.NIL \mid k \leqslant n\} + \sum \{c_k.\Omega \mid k \geqslant n\}.$$

Then $P_0, ..., P_n, ...$ is a chain and it is natural to expect that its limit would be P, the denotation of

$$\sum \{c_k.NIL \mid k \geqslant 0\}.$$

However, in the proposed model, $(P_n \mid c!1.NIL)\backslash c = \Omega$ for every n, whereas $(P \mid c!1.NIL)\backslash c = NIL$.

In other words, accommodating any aspect of unbounded nondeterminism within the testing or failures framework is nontrivial and best avoided if at all possible. Attempts in this direction may be seen in [He87] and the previously cited [Ros88b]. So a major contribution of the present paper is to show that value-passing languages, which a priori exhibit forms of unbounded nondeterminism, can be accurately modeled using standard mathematical constructions which are usually only considered adequate for finitely branching processes.

In [Ros88a] a denotational model for OCCAM, ([In84]) is presented and in [HR88] equational laws for this model are investigated. This work has much in common with our research, although their language is imperative where ours is applicative. Moreover, in this work there is the underlying assumption that the set of values passed between processes is finite. This is necessary because even in the standard failures model of [Hoa85], which is the underlying basis for these papers, hiding with respect to an infinite number of values is not continuous; the example above may be used to show this. Their model also lacks a behavioural characterization; this poses the problem of how to develop a theory of testing for imperative languages which contain assignment statements.

ACKNOWLEDGMENTS

The first author acknowledges the support SERC and the ESPRIT/BRA project Concur, while the second author was funded by Århus/Aalborg University.

RECEIVED August 14, 1991; FINAL MANUSCRIPT RECEIVED August 19, 1991

REFERENCES

[Bri86] BRINKSMA, E. (1986), A turotial on LOTOS, *in* "Proceedings of IFIP Workshop on Protocol Specification, Testing and Verification V" (M. Diaz, Ed.), pp. 73–84, North-Holland, Amsterdam.

[DNH84] DENICOLA, R., AND HENNESSY, M. (1984), Testing equivalences for processes, *Theoret. Comput. Sci.* **24**, 83–113.

[Gue81] GUESSARIAN, I. (1981), "Algebraic Semantics," Lecture Notes in Computer Science, Vol. 99, Springer-Verlag, Berlin/New York.

[HP80] HENNESSY, M., AND PLOTKIN, G. (1980), "A Term Model for CCS," Lecture Notes in Computer Science, Vol. 88, Springer-Verlag, Berlin/New York.

[He85] HENNESSY, M. (1985), Acceptance trees, *J. Assoc. Comput. Mach.* **32**, No. 4, 896–928.

[He87] HENNESSY, M. (1987), An algebraic theory of fair asynchronous communicating processes, *Theoret. Comput. Sci.* **49**, 121–143.

[He88] HENNESSY, M. (1988), "Algebraic Theory of Processes," MIT Press, Cambridge, MA.

[He89] HENNESSY, M. (1989), "A proof system for communicating processes with value-passing, *in* "Proceedings of Foundations of Software Technology and Theoretical Computer Science, Ninth Conference, Bangalore," pp. 325–339, Lecture Notes in Computer Science, Vol. 405, Springer-Verlag, Berlin/New York; to be published in *Formal Aspects Comput.*

[Hoa85] HOARE, C. A. R. (1985), "Communicating Sequential Processes," Prentice-Hall International, London.

[HR88] HOARE, C. A. R., AND ROSCOE, A. W. (1988), The laws of Occam, *Theoret. Comput. Sci.* **60**, 177–229.

[In84] INMOS LTD. (1984), "The Occam Programming Manual," Prentice–Hall, London.

[Miln88] MILNE, R. (1988), "Concurrency Models and Axioms," RAISE/STC/REM/6/V2, STC Technology.

[Mil80] MILNER, R. (1980), "A Calculus of Communicating Systems," Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, Berlin.

[Mil89] MILNER, R. (1989), "Calculus for Communication and Concurrency," Prentice–Hall, London.

[Plo81] PLOTKIN, G. (1981), "Lecture Notes in Domain Theory", Univ. of Edinburgh.

[Ros88a] ROSCOE, A. W. (1988), "Denotational Semantics for Occam," PRG Monograph, Oxford University.

[Ros88b] ROSCOE, B. (1988), "Two Papers on CSP," Oxford University Technical Report PRG-67.

[Smt86] SCHMIDT, D. (1986), "Denotational Semantics," Alley & Bacon, Rocklejih, NJ.

[SP82] SMYTH, M., AND PLOTKIN, G. (1982), The category-theoretic solution of recursive domain equations, *SIAM J. Comput.* **11**, No. 4.