A Model for the π- Calculus *

M. Hennessy
University of Sussex

Abstract

We develop a semantic theory based on testing for a minor variant of the π-calculus. The resulting semantic equivalence can be characterised using of acceptance sets and can also be characterised as an equational theory. We define a class of interpretations for the π-calculus and construct one which is *fully-abstract*. Moreover the interpretation we construct is initial in the class of all fully-abstract interpretations.

*This work has been supported by the ESPRIT/BRA CONCUR project

1 Introduction

In [MPW92a], [MPW92b], a calculus of mobile processes, the π-calculus, is presented. The first reference is an introduction to the calculus and the second develops a semantic theory based on bisimulations, [Mil89]. The π-calculus is an extension of the process algebra *CCS*, a more primitive calculus for describing and manipulating processes which perform uninterpreted actions. In the π-calculus these actions are now interpreted as either the input or output of values along channels. The power of the extension comes from the fact that the values sent and received are themselves channel names and moreover the language allows the transmission of private channels between processes.

In [HI91] a similar extension to a *CCS*-like language is developed but here the values transmitted are simple data-values such as the integers. Although processes may test these data-values and base their future behaviour on the outcome, unlike the π-calculus, these values can not change radically the communication topology of processes. The theory of this extension, called *VPL*, is based on the testing approach to processes, as presented in [Hen88]. This is in terms of a preorder $\sqsubseteq$ and roughly speaking $p \sqsubseteq q$ for two processes p and q if q guarantees all tests guaranteed by p. In [HI91] a sound and complete proof system is given for the language *VPL* and a fully-abstract model, based on the acceptance trees of [Hen88] is also defined.

In this paper we show that a similar theory may be developed for mobile processes. As usual we modify the language somewhat, the main difference being that the *CCS* operator for internal moves τ is replaced by a binary internal choice operator, $\oplus$. We call the modified language L_π and apart from this modification it is more or less the same as the π-calculus. A testing preorder is developed for this language and we show that it may be characterised in terms of acceptance trees. We also give an algebraic characterisation of the preorder. The equations involved are relatively straightforward. They partition naturally into two sets. The first consist of standard testing equations, those from [Hen88] together with an extra equation, originally given in [HI91], which is required because of the interpreted nature of the actions – as input and output of values along channels. The second set is a subset of the equations from [MPW92b] for the π-calculus and are mainly concerned with the restriction operator; this is not surprising as the real power of the π-calculus stems from restriction.

Finally we discuss the problem of giving a denotational interpretation for the language. We first define an appropriate class of interpretations and show that there exists a fully-abstract one, i.e. an interpretation which identifies and only identifies processes which can not be distinguished by tests. Unfortunately the construction of this interpretation is as a term model, built from the syntax of the language using a provability relation. It would be more interesting if a representation could be given in terms of, for example, the acceptance trees of [Hen88]. But for the moment we have been unable to do this. The major obstacle is with the restriction operator. It is far from obvious how to adapt the definition of acceptance trees so that they can support a reasonable interpretation of restriction.

As far as the author is aware all suggested theories in the literature for the π-calculus are based on bisimulation equivalence apart from the work reported in [BD92]. Here a testing theory for the finite terms of the π-calculus is defined and equationally characterised. This work is quite similar to ours although the authors work with the exact

syntax of the language of [MPW92a] whereas we have modified it slightly so as to obtain a marginally cleaner equational characterisation. But the main difference between the present paper and [BD92] is that we are primarily interested in developing a denotational model of the language and have used the equational characterisation as a tool in its construction.

2 The Language

In this section we define the language, called L_π, and its operational semantics. The language is a minor variant of the π-calculus as presented in [MPW92a],[MPW92b], the main difference being that two nondeterministic operators, *internal nondeterministic choice* and *external nondeterministic choice* are used in place of the original nondeterministic operators from *CCS*. The π-calculus is a particular instance of a family of an algebraic description languages for communicating processes. The processes use communication channels to exchange values and in the π-calculus these values are themselves communication channels. The real power of the language comes from the fact that *private* channels may be exchanged between processes. The reader is referred to [MPW92a] for a thorough introduction to the language. We limit ourselves to a rather terse exposition of our variant and its operation semantics. This in turn is very similar to the exposition of the original π-calculus in [MPW92b].

Let PV be a set of process variables, ranged over by X, Y, and $\mathcal{N}$ a countable set of channel names, ranged over by x, y. The set of process terms, ranged over by $t, u, \ldots$ is then defined by the following grammar:

$$
\begin{array}{rcl}
t & ::= & \Omega \mid nil \\
 & & \mid t + t \mid t \oplus t \\
 & & \mid pre \\
 & & \mid t \mid t \\
 & & \mid (x)t \\
 & & \mid \ if\, be\, then\, t\, else\, u \\
 & & \mid X \mid rec\, X.\, t \\
pre & ::= & \overline{x}y \mid x(y) \\
be & ::= & x = y \mid \neg be \mid be \vee be \mid be \wedge be
\end{array}
$$

Here Ω represents the completely undefined process and *nil* the process which has terminated. Next we have the two types of nondeterminism, $+$ representing external nondeterminism and $\oplus$ the internal form. Input and output along channels are represented by $x(y).t$ and $\overline{x}y.t$ respectively. The term $x(y).t$ represents a process which may input a channel name from the channel x, "bind" that channel name to the name y in the process t. On the other hand $\overline{x}y.t$ may output the name y along the channel name x and proceed like the process t. The term $t \mid u$ represents the process consisting of two subprocesses t and u running in parallel while in $(x)t$ all occurrences of the channel name x are local. This is the representation in the π-calculus of the restriction operator of *CCS*, [Mil89],

where it is written as $t \setminus x$. In fact it is with this operator that the communication of private channels may be represented in the language. Unlike the original π-calculus we use an *if ... then ... else ...* statement. The process *if be then t else u* acts like t if the boolean expression *be* is true and like u otherwise. We allow a very simple language of boolean expressions. Essentially it allows the testing of identity between channel names. Finally we have recursive definitions; intuitively the process $rec\, X.\ t$ is equivalent to a process X where X has been defined by the equation $X = t$. We use Σ to denote the set of operators $\{nil, \Omega, (x), \overline{x}y, +, \oplus, |\}$, i.e. all the operators except prefixing by input actions; the interpretation of this last operator will require special attention.

As usual rec acts like a binder for process variables and we have an appropriate form of substitution: $t[u/X]$ is the term which results from substituting u for all free occurrences of X in t where the substitution is defined so that free variables are not captured. We are mainly interested in *closed terms*, i. e. terms with no free occurrences of process variables. We use $\mathcal{P}$ to denote the set of such terms, which we refer to as *processes*, and individual processes are referred to using the meta-variables $p, q, \ldots$.

More importantly we have two binders for channel names. In $(x)t$ all occurrences of the name x in t are bound and in $x(y).t$ all occurrences of y are also bound. These binders give rise in the normal way to the definition of free and bound names which occur in a process, $fn(t), bn(t)$ respectively. These functions will also be liberally applied to sets of terms, sequences of terms etc. . We will also use $n(t)$ to refer to the union of these two sets, i. e. the set of names which occur in t. When substituting names in processes we have to be careful to preserve their free names. In general a *substitution* is a function σ from $\mathcal{N}$ to $\mathcal{N}$ which is almost everywhere the identity. We use $t\sigma$ to denote the process which results from simulataneously substituting in t all free occurrences of x by $\sigma(x)$, with change of bound names to avoid captures. A precise definition, for a somewhat different language, may be found in [Sto88]. We use the normal notation for substitutions when convenient, writing $\{y_1/x_1, \ldots, y_n/x_n\}$ for the substitution σ whose non-trivial domain is $\{x_1, \ldots, x_n\}$ and is defined by $\sigma(x_i) = y_i$. We also use the standard notation for modifying substitutions, $\sigma[x \mapsto y]$ being the substitution which is the same as σ except that x is mapped to y. The relation of *alpha–convertability* on processes, defined in the normal way, is denoted by $\equiv_\alpha$ whereas $\equiv$ will mean syntactic identity. We use without comment standard properties of substitution and $\equiv_\alpha$ and sometimes we rely on specific properties of the actual definition of substitution from [Sto88]. One useful property of this definition is that if σ and σ' agree on $fn(t)$ then $t\sigma \equiv_\alpha t\sigma'$.

We formalise our intuitions about the behaviour of processes by giving an operational semantics to $\mathcal{P}$. As in [Hen88] this consists of a relation $\rightarrowtail$ which describes the possible internal actions, such as internal communication, between processes and a collection of next state relations $\xrightarrow{a}$ where a ranges over the possible external actions processes can perform. These external actions are of two froms, *input* and *output*, but because of the particular values being exchanged the situation is somewhat more complicated than for the operational semantics of the value-passing version of standard *CCS*. There are three kinds of external actions.

1. A *free output* action. Here $p \xrightarrow{\overline{x}y} q$ implies that the process p can output the free name y to the channel x or on the port $\overline{x}$. A typical example of such an action arises from the process $\overline{x}y.p$ and according to the definition of the transition system we will have $\overline{x}y.p \xrightarrow{\overline{x}y} p$.

2. An *input* action $x(y)$. Intuitively $p \xrightarrow{x(y)} q$ means that the process p may recieve *any* name v from the channel x and thereby be transformed into $q\{v/y\}$. It is important to realise that this is unlike the standard input actions of say [HI91] or [Mil89]. Here y is being used as a *place holder* or *reference* and the $p \xrightarrow{x(y)} q$ does not mean that p has actually input the name y on the channel x and been transformed to q. Instead it simply encodes the ability of p to input from the channel x and the resulting effect. Processes of the form $x(y).p$ give rise to input actions.

3. A *bound output* action $\overline{x}(y)$. Here $p \xrightarrow{\overline{x}(y)} q$ means that p may output a private name along the channel x but once more y is not necessarily this private name. As in the previous case it represents a reference (in q) to this private channel. It is these kinds of actions, which do not appear in standard *CCS*, which gives the language its power. They are typically performed by processes of the form $(y)\overline{x}.p$. The reader is referred to [MPW92a] for a more detailed discussion.

As in [MPW92b] the relevant properties of these external actions are given in the following table:

a	Kind	Free/Bound	fn(a)	bn(a)	Subject	Object
$\overline{x}y$	Free Output	f	$\{x, y\}$	$\emptyset$	$\overline{x}$	y
$x(y)$	Input	b	$\{x\}$	$\{y\}$	x	y
$\overline{x}(y)$	Bound Output	b	$\{x\}$	$\{y\}$	$\overline{x}$	y

We use *Act* to denote the set of these actions. As can be seen from the table all the actions occurring in a free action are free whereas in a bound action the object is bound while the subject is free.

The defining rules for these transitions are given in Figures 1 and 2. These are a mixture of the usual rules for internal actions taken from [Hen88] and those for the π-calculus from [MPW92a]. The latter are the most complicated but rather than explaining them in detail we refer the reader to their detailed exposition in [MPW92a] and [MPW92b]. Note that in the rules Sum, Par, Choice, Com and Close we have omitted the symmetrical versions. In the two If rules we have also assumed some standard evaluation mechanism for Boolean expressions, with $[\![be]\!]$ always evaluating to either true or false.

As an example of the operational semantics consider the process $p \mid q \mid r$ where p, q, r are $x_1(y).\overline{x}_2y.p'$, $(z)\overline{x}_1z.z(y).q'$ and $x_2(y).\overline{y}x.r'$ respectively. Then q can send a private channel name to p along the channel x_1 who in turn can send it to r using x_2 and finally r can use this private name to send x to q. The resulting state is then $(v)(p' \mid q'\{x/y\} \mid r')$ where v is represents the private channel and does not occur in $p \mid q \mid r$.
The first move in this computation is

$$p \mid q \mid r \rightarrowtail (v)(\overline{x}_2v.p' \mid v(y).q') \mid r$$

and its derivation uses the rules Open, Output, Input, Close and Par. The next move is

$$(v)(\overline{x}_2v.p' \mid v(y).q') \mid r \rightarrowtail (v)(p' \mid v(y).q' \mid \overline{v}x.r')$$

and this uses the same rules. The final move is then

$$(v)(p' \mid v(y).q' \mid \overline{v}x.r') \rightarrowtail (v)(p' \mid q'\{x/y\} \mid r').$$

[t]

Input $\dfrac{-}{x(y).p \xrightarrow{x(v)} p\{v/y\}} \quad v \notin fn((y)p)$

Output $\dfrac{-}{\overline{x}y.p \xrightarrow{\overline{x}y} p}$

Sum $\dfrac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'}$

Par $\dfrac{p \xrightarrow{a} p'}{p \mid q \xrightarrow{a} p' \mid q} \quad bn(a) \cap fn(q) = \emptyset$

Res $\dfrac{p \xrightarrow{a} p'}{(y).p \xrightarrow{a} (y).p'} \quad y \notin n(a)$

Open $\dfrac{p \xrightarrow{\overline{x}y} p'}{(y).p \xrightarrow{\overline{x}(v)} p'\{v/y\}} \quad y \neq x \text{ and } v \notin fn((y)p')$

If $\dfrac{p \xrightarrow{a} p', \quad [\![be]\!] = tt}{if\ be\ then\ p\ else\ q \xrightarrow{a} p'}$

$\dfrac{q \xrightarrow{a} q', \quad [\![be]\!] = ff}{if\ be\ then\ p\ else\ q \xrightarrow{a} q'}$

Figure 1: Rules for external actions

In the remainder of this section we develop some technical results about this transition system and it may be skipped by the reader uninterested in technical details. Most of the proofs are omitted as they are rather tedious and usually proceed by syntactic analysis. The first two results are taken directly from [MPW92b] and the proofs can be transferred directly to our language.

Lemma 2.1 *If* $p \succ\!\!\rightarrow p'$ *then* $fn(p') \subseteq fn(p)$ *and if* $p \xrightarrow{a} p'$ *then* $fn(p') \subseteq fn(p) \cup bn(a)$ *and* $fn(a) \subseteq fn(p)$ □

Definition 2.2 We use the phrase

if $p \xrightarrow{a} p'$ then *equally* $q \xrightarrow{a} q'$

to mean that if $p \xrightarrow{a} p'$ may be inferred from the transition rules then $q \xrightarrow{a} q'$ may also be inferred, with an inference of no greater depth. We use a similar notation for internal arrows. □

Lemma 2.3 *Suppose that* $p \xrightarrow{\alpha(y)} p'$ *where* $\alpha = x$ *or* $\alpha = \overline{x}$ *and that* $z \notin n(p)$. *Then equally* $p \xrightarrow{\alpha(z)} p''$ *for some* p'' *such that* $p'' \equiv_\alpha p'\{z/y\}$. □

[t]

Ω $\quad \dfrac{-}{\Omega \rightarrowtail \Omega}$

Rec $\quad \dfrac{-}{rec\,X.\ t \rightarrowtail t[rec\,X.\ t/X]}$

Choice $\quad \dfrac{p \rightarrowtail p'}{p \oplus q \rightarrowtail p'}$

Op $\quad \dfrac{p_i \rightarrowtail p_i'}{op(\ldots, p_i, \ldots) \rightarrowtail op(\ldots, p_i', \ldots)}$ for $op \in \{+, |, (y)\}$

Com $\quad \dfrac{p \xrightarrow{\overline{x}y} p', \quad q \xrightarrow{x(z)} q'}{p \mid q \rightarrowtail p' \mid q'\{y/z\}}$

Close $\quad \dfrac{p \xrightarrow{\overline{x}(w)} p', \quad q \xrightarrow{x(w)} q'}{p \mid q \rightarrowtail (w)(p' \mid q')}$

If $\quad \dfrac{\begin{array}{c} p \rightarrowtail p', \quad [\![be]\!] = tt \\ \hline if\ be\ then\ p\ else\ q \rightarrowtail p' \end{array}}{\begin{array}{c} q \rightarrowtail q', \quad [\![be]\!] = ff \\ \hline if\ be\ then\ p\ else\ q \rightarrowtail q' \end{array}}$

Figure 2: Rules for internal transitions

In the operational semantics bound names are liberally renamed and it is important to establish that α-conversion does not seriously affect the behavioural properties of processes. The next series of results have this in mind. In the next section a new kind of action, a *free input* action, will be seen to be important and therefore these results will be also established for these actions. Recall that $p \xrightarrow{x(y)} p'$ means that y is acting as a reference in p' to where names which are received by p are placed. This is actually how it is used in the rule Com of Figure 2. So it is natural to define a free input action by

$$p \xrightarrow{xy} p'\{y/z\} \text{ whenever } p \xrightarrow{x(z)} q.$$

We use $EAct$, to denote this extended set of Actions, i.e.

$$EAct = \{\, xy, x(y), \overline{x}y, \overline{x}(y) \mid x, y \in \mathcal{N} \,\}.$$

Now we will use a to range over this augmented set of actions.
In order to state these results it is convenient to extend α-conversion to actions in $EAct$ and more generally sequences over $EAct^*$. The most convenient way of doing this is to view sequences as very simple sequential processes and apply the standard definition which already exists for processes. Thus if a is a free action and $a \equiv_\alpha a'$ then necessarily a' must be a but $x(y) \equiv_\alpha x(z)$ for any pair of names y and z. Similarly if $\overline{x}(y).s \equiv_\alpha u$

then u must be of the form $\overline{x}(z).u'$ for some z such that $s \equiv_\alpha u'\{y/z\}$. We will make frequent use of the notion of a *harmless* action or sequence of actions. In a particular statement a sequence of actions is considered harmless if the bound variables are different than any of the variables appearing in the rest of the statement. This definition is of course context dependent but in each context the meaning will be perfectly clear.

Lemma 2.4 *For any $a \in EAct$*

1. *$p \xrightarrow{a} p'$ implies that for any harmless a' such that $\sigma(a) \equiv_\alpha \sigma(a')$ $p\sigma \xrightarrow{a'} r$ for some r such that $r \equiv_\alpha p'\sigma[bn(a) \mapsto bn(a')]$*

2. *$p \rightarrowtail p'$ implies $p\sigma \rightarrowtail r$ for some r such that $r \equiv_\alpha p'\sigma$.*

Proof: The two statements are proved simultaneously by induction on the derivations. In the first statement we mean that a' should be such that $bn(a') \cap n(p, a) = \emptyset$. This will be true of almost all a' such that $a \equiv_\alpha a'$ because the set of names appearing in p and a is finite. □

The converse is not true in general but we can obtain a partial result by restricting the allowed substitutions to be injective. However the statement of the result is complicated by the presence of free input actions. If $p\sigma$ performs the action $x'y'$ then there is an x such that $\sigma(x) = x'$ but there may not be a y such that $\sigma(y) = y'$.

Lemma 2.5 *Let σ be injective on $fn(p)$ and $a \in EAct$.*

1. *$p\sigma \xrightarrow{a'} r$ implies that for any harmless a and substitution σ' such that $p\sigma \equiv_\alpha p\sigma'$ and $\sigma'(a) \equiv_\alpha a'$ there exists a p'such that $p \xrightarrow{a} p'$ and $r \equiv_\alpha p'\sigma'[bn(a) \mapsto bn(a')]$.*

2. *$p\sigma \rightarrowtail r$ implies $p \rightarrowtail p'$ for some p' such that $p'\sigma \equiv_\alpha r$.*

Proof: Once more both of these are proved simultaneously by induction on the derivations. □

These may now be combined to give the first result about α-conversion. If $p \equiv_\alpha q$ and p does an internal move to p' then q can also do an internal move to some q' such that $p' \equiv_\alpha q'$. But this is not true for external action because the rule Open can transform a bound variable into a free variable. So when matching external moves from α-equivalent processes a record must be kept of these newly generated free variables.

Proposition 2.6 *If $p \equiv_\alpha q$ then*

1. *$p \xrightarrow{a} p'$ implies that for any harmless a' such that $a \equiv_\alpha a'$ there exists a q' such that $q \xrightarrow{a} q'$, $p' \equiv_\alpha q'\{bn(a)/bn(a')\}$ and $q' \equiv_\alpha p'\{bn(a')/bn(a)\}$*

2. *$p \rightarrowtail p'$ implies that, for some q', $p \rightarrowtail q'$ and $p' \equiv_\alpha q'$.*

Proof: Once more both results are proved simultaneously by induction on the length of the derivations. Note that in i) it is sufficient to prove $p' \equiv_\alpha q'\{bn(a)/bn(a')\}$ only and this only in the case where $bn(a) \notin n(p,q)$. For suppose we have shown that this is true. Then let $p \xrightarrow{a} p'$ for an arbitrary a. There exists an a'' such that $a \equiv_\alpha a''$ and $bn(a) \notin n(p,q)$. Applying Lemma 2.3 if necessary we can find a p'' such that $p \xrightarrow{a''} p''$ and $p'' \equiv_\alpha p'\{bn(a'')/bn(a)\}$. From our assumption we then obtain, for any harmless a' such that $a' \equiv_\alpha a'' \equiv_\alpha a$, a q' such that $q \xrightarrow{a'} q'$ and $p'' \equiv_\alpha q'\{bn(a'')/bn(a')\}$. However now one can check that

$$p' \equiv_\alpha p'\{bn(a'')/bn(a)\}\{bn(a)/bn(a'')\} \equiv_\alpha q'\{bn(a)/bn(a')\}$$

and

$$q' \equiv_\alpha q'\{bn(a'')/bn(a')\}\{bn(a')/bn(a'')\} \equiv_\alpha p'\{bn(a')/bn(a)\}.$$

□

These results can be extended to sequences of actions. For $s \in EAct^*$ let the relation $\stackrel{s}{\Longrightarrow}$ be defined in the standard way:

1. $p \stackrel{\varepsilon}{\Longrightarrow} p$
2. $p \rightarrowtail p''$, $p'' \stackrel{s}{\Longrightarrow} p'$ implies $p \stackrel{s}{\Longrightarrow} p'$
3. $p \xrightarrow{a} p''$, $p'' \stackrel{s}{\Longrightarrow} p'$ implies $p \stackrel{a.s}{\Longrightarrow} p'$.

We use $\underline{bn}(s)$ to denote the sequence of bound names which appear in s and by an abuse of notation $\sigma[\underline{bn}(s) \mapsto \underline{bn}(s')]$ is used to denote the substitution $\sigma[x_1 \mapsto y_1, \ldots, x_k \mapsto y_k]$ where $x_1, \ldots, x_k$, and $y_1, \ldots, y_k$ are the sequences $\underline{bn}(s), \underline{bn}(s')$ respectively. For this to make sense s and s' must have the same number of bound variables.

Proposition 2.7 *For any $s \in EAct^*$*

1. *$p \stackrel{s}{\Longrightarrow} p'$ implies that for any harmless s' such that $\sigma(s) \equiv_\alpha s'$ there exists an r such that $p\sigma \stackrel{s'}{\Longrightarrow} r$ and $r \equiv_\alpha p'\sigma[\underline{bn}(s) \mapsto \underline{bn}(s')]$*
2. *if σ is injective on $fn(p)$ then $p'\sigma \stackrel{s'}{\Longrightarrow} r$ implies that for any harmless s and substitution σ' such that $p\sigma \equiv_\alpha p\sigma'$ and $s' \equiv_\alpha \sigma'(s)$ there exists an p' such that $p \stackrel{s}{\Longrightarrow} p'$ and $r \equiv_\alpha p'\sigma'[\underline{bn}(s) \mapsto \underline{bn}(s')]$*
3. *if $p \equiv_\alpha q$ and $p \stackrel{s}{\Longrightarrow} p'$ then, for any harmless s' such that $s \equiv_\alpha s'$, $q \stackrel{s'}{\Longrightarrow} q'$ for some q' such that $p' \equiv_\alpha q'\{\underline{bn}(s)/\underline{bn}(s')\}$ and $q' \equiv_\alpha p'\{\underline{bn}(s')/\underline{bn}(s)\}$.*

Proof: The three statements are proved simultaneously by induction on the length of the derivation of $\stackrel{s}{\Longrightarrow}$. For each of the statements one of the three previous results are employed when an individual move of the derivation is being considered. □

This completes the analysis of α-conversion and in the remainder of this section we show that the possible computations from processes are in some sense finite branching. For any action $a \in Act$ let $D(p,a) = \{\, p' \mid p \xrightarrow{a} p' \,\}$.

Lemma 2.8 *For any a, p the set $D(p,a)$ is finite.*

Proof: The proof is straightforward by structural induction on p. Note that if p has the form $recX.\ t$ then this set is empty for every a. □

Of course a process may be able to perform an infinite number of different actions. For example $x(y).y!z.nil$ can perform the action $x(v)$ for any v and $(y)\overline{x}(y).nil$ can perform $\overline{x}(v)$ for any v. But the number of names which may be used as subjects is finite as is the number of free names which may be output. Let $Subj(p)$ denote the set of names x such that $p \stackrel{a}{\longrightarrow} p'$ for some p' and a such that x is the subject of a and let $OutN(p)$ be the set of y such that for some p' and x $p \stackrel{\overline{x}y}{\longrightarrow} p'$.

Lemma 2.9 *For every p the sets $Subj(p)$ and $OutN(p)$ are finite.*

Proof: Again a simple proof by structural induction. □

However the use of bound variables also means that the set of states to which a process may evolve by an internal move may not be finite. However it is up to α-conversion. Let $ID(p) = \{\, [p'] \mid p \rightarrowtail p' \,\}$ where $[p]$ denotes the equivalence class of p with respect to $\equiv_\alpha$.

Lemma 2.10 *For every p the set $ID(p)$ is finite.*

Proof: By structural induction on p. Note that the case when p has the form $recX.t$ the result is trivial since $ID(recX.t) = \{t[recX.t/X]\}$. The only non-trivial case is when t has the form $p \mid q$. If $p \mid q \rightarrowtail r$ then r must be of the form $p' \mid q'$ and there are three possible ways in which this move can be inferred:

1. using the rule Op in which case $p \rightarrowtail p'$ and q' is q or $q \rightarrowtail q'$ and p' is p. By induction $ID(p)$ and $ID(q)$ are finite and therefore there are only finite number of $[r]$ which can be inferred from this rule.

2. using the rule Com. Here either $p \stackrel{\overline{x}y}{\longrightarrow} p'$ and $q \stackrel{x(z)}{\longrightarrow} q''$ with $q' = q''\{y/z\}$ or symmetrically $q \stackrel{\overline{x}y}{\longrightarrow} q'$ and $p \stackrel{x(z)}{\longrightarrow} p''$ with $p' = p''\{y/z\}$. It is sufficient to consider the first case where we show that there is only a finite number of ways of using Com in this way. Pick some v which does not appear in $p \mid q$. By Lemma 2.3 we know that $q \stackrel{x(v)}{\longrightarrow} q'''$ such that $q''' \equiv_\alpha q''\{v/z\}$. Therefore each $p' \mid q'$ inferred in this way must be such that $p' \in D(p, \overline{x}y)$ and $q' \equiv_\alpha q'''\{y/v\}$ where $q''' \in D(q', x(v))$. Since both of these sets are finite there can only be a finite number of such $p' \mid q'$ up to α-equivalence.

3. using the rule Close. This is treated exactly as in the previous case.

□

3 Testing Processes

Given the operational semantics of the previous section we may now apply the standard theory of testing as developed in [Hen88]. To this end we assume a special name called ω which is used to denote success. A *test* or *experiment* is then simply a process which may use this extra name and applying a test e to a process p consists in running the process $e \mid p$ to completion. A *computation* from $e \mid p$ is a complete sequence of the form

$$e \mid p = s_0 \rightarrowtail s_1 \rightarrowtail \ldots \rightarrowtail s_k \rightarrowtail \ldots$$

i.e. it is either infinite or if s_n is the last element there must be no s' such that $s_n \rightarrowtail s'$. Such a computation is *successful* if some s_k can report success, i. e. $s_k \xrightarrow{\overline{\omega}(x)}$ for any name x. We often write this as $s_k \in Succ$. Then we write

$$p \; must \; e$$

if every computation from $p \mid e$ is successful. Finally we say

$$p \sqsubseteq_{\approx} q$$

if for every experiment e, p *must* e implies q *must* e. We use $\eqsim$ to denote the kernel of $\sqsubseteq_{\approx}$ and is obviously an equivalence relation.

We wish to establish two results about testing, firstly that the preorder is preserved by all the operators of the language and secondly that for any test e if p *must* e then for some finite approximation d to p, p *must* e. The latter is easier if we have developed an equational theory for the language and this is the topic of the next section. The proof of the latter is more straightforward in terms of an alternative characterisation of $\sqsubseteq_{\approx}$.

Recall from [Hen88] that for the pure version of the language this alternative characterisation is in terms of *acceptance sets* of actions which code up the possible next moves a process can perform after having executed a sequence of actions. This characterisation will not apply directly here because some of the transitions which processes can perform record their potential rather than actual moves. For example let p, q denote $x(y).\overline{x}y.nil$ and $x(y).$ *if ff then* $\overline{z}y.nil$ *else* $\overline{x}y.nil$ respectively. Then obviously $p \eqsim q$ but the set of input actions they can perform are different; p can perform the action $x(z)$ which q can not perform. To overcome this problem we use the new kind of action, a *free input action*, introduced in the previous section. These free input actions give us all the behavioural information we require of the ability of processes to input and therefore we need not consider bound input actions at all.

Let $RAct$ denote the set of actions $\{ xy, \overline{x}y, x(y) \mid x, y \in \mathcal{N} \}$. The alternative characterisation of $\sqsubseteq_{\approx}$ is in terms of the ability of processes to perform actions from this set. Note that this still contains bound output actions and these continue to present problems. For example $(y).\overline{x}y.y(z).nil$ and $(w).\overline{x}w.y(w).nil$ are obviously behaviourally equivalent but they offer different possibilities with respect to the actions from $RAct$. The latter can perform the action $\overline{x}(w)$ which the former can not; but it can perform $\overline{x}(y)$ which is equivalent if we only consider the free names involved. We overcome this problem by working only with sequences from $RAct^*$ where the bound variables are *new*, i. e. do not occur in the processes under consideration.

First let us define *convergence* with respect to sequences of actions.

Definition 3.1 For every sequence $s \in RAct$

1. let $\downarrow s$ be defined by
   - $p \downarrow \varepsilon$ if there is no infinite internal computation from p, i.e. no infinite computation of the form $p \succ\!\!\rightarrow p_1 \succ\!\!\rightarrow \ldots \succ\!\!\rightarrow p_k \succ\!\!\rightarrow \ldots$.
   - $p \downarrow a.s$ if $p \downarrow$ and for every p' such that $p \stackrel{a}{\Longrightarrow} p'$ $p' \downarrow s$.
2. let $p \Downarrow s$ if for every s' such that $s \equiv_\alpha s'$ $p \downarrow s'$.

□

To prove that the latter is preserved by α-conversion we need a lemma.

Lemma 3.2 *For every process p*

1. *$p\sigma \downarrow$ implies $p \downarrow$*
2. *if σ is injective on $fn(p)$ then $p \downarrow$ implies $p\sigma \downarrow$*
3. *$p \equiv_\alpha q$ and $p \downarrow$ implies $q \downarrow$.*

Proof: As an example we prove the second statement. Suppose

$$p\sigma \succ\!\!\rightarrow r_1 \succ\!\!\rightarrow \ldots \succ\!\!\rightarrow r_n \succ\!\!\rightarrow \ldots$$

is an infinite sequence. By Lemma 2.5 $p \succ\!\!\rightarrow p_1$ such that $p_1\sigma \equiv_\alpha r_1$ and by Proposition 2.7, part (3), $p_1\sigma \succ\!\!\rightarrow r_2'$ such that $r_2 \equiv_\alpha r_2'$. Again by Lemma 2.5 $p_1 \succ\!\!\rightarrow p_2$ such that $p_2\sigma \equiv_\alpha r_2$. Continuing in this way we obtain an infinite internal computation from p. □

As a corollary we have

Proposition 3.3

1. *If $p \equiv_\alpha q$ and $p \Downarrow s$ then $q \Downarrow s$.*
2. *if σ is injective on $fn(p)$ then $p \Downarrow s$ implies $p\sigma \Downarrow \sigma(s)$*
3. *if $p\sigma \Downarrow s'$ then for every σ' and s such that $p\sigma \equiv_\alpha p\sigma'$ and $\sigma'(s) \equiv_\alpha s'$, $p \Downarrow s$.*

Proof: The first statement directly from the previous lemma and the results at the end of the previous section. For suppose $p \Uparrow s$, i. e. for some prefix s' of some s'' such that $s \equiv_\alpha s''$ $p \stackrel{s'}{\Longrightarrow} p'$ such that $p \uparrow$. So by Proposition 2.7 $q \stackrel{s_1'}{\Longrightarrow} q'$ for some q' and s_1' such that $s' \equiv_\alpha s_1'$ and $q' \equiv_\alpha p'\{\underline{bn}(s')/\underline{bn}(s)\}$. Since $p' \uparrow$ it follows from the previous lemma that $p'\{\underline{bn}(s')/\underline{bn}(s)\} \uparrow$ and therefore that $q' \uparrow$. This in turn implies that $q \Uparrow s$.

We leave the second statement to the reader and concentrate on the last. Suppose $p \Uparrow s$, i. e. $p \stackrel{u}{\Longrightarrow} p'$ where $p \uparrow$ for some prefix u of s. By Proposition 2.7 this means that $p\sigma' \stackrel{u'}{\Longrightarrow} r$ for some r and u' such that $r \equiv_\alpha p'\sigma'[\underline{bn}(u) \mapsto \underline{bn}(u')]$ and $u' \equiv_\alpha \sigma'(u)$. From

the previous lemma this means that $r \uparrow$ and therefore $p\sigma' \Uparrow u'$, i. e. $p\sigma' \Uparrow s'$. Applying clause (1) we obtain $p\sigma \Uparrow s$. □

In fact in order to establish that a process is convergent with respect to all sequences which are α-equivalent to s it is sufficient to establish it for a sequence whose bound variables are new, i.e. a harmless sequence.

Lemma 3.4 *If $bn(s) \cap fn(p) = \emptyset$ then $p \Downarrow s$ if and only if $p \downarrow s$.*

Proof: One direction is immediate. So suppose that $p \downarrow s$, $s \equiv_\alpha s'$ and $p \stackrel{s'}{\Longrightarrow} p'$. We need to show that $p' \downarrow$. By Proposition 2.7 we know $p \stackrel{s}{\Longrightarrow} p_1$ such that $p' \equiv_\alpha p_1\{\underline{bn}(s)/\underline{bn}(s')\}$. Now $p_1 \downarrow$ and because $\underline{bn}(s)$ are new the substitution is injective on $fn(p_1)$ and therefore by the Lemma 3.2 $p_1\{\underline{bn}(s)/\underline{bn}(s')\} \downarrow$ which in turn means $p' \downarrow$. □

We now define the possible acceptance sets of a process after a sequence of actions s. The definition is standard but the acceptance sets of processes will be compared with respect to a restricted set of sequences. We say p is *stable* if $p \not\rightarrowtail p'$ for no p'. Let

$$\mathcal{A}(p,s) = \{\, Subj(p') \mid p \stackrel{s}{\Longrightarrow} p', p' \ \ stable \,\}$$

Lemma 3.5 *For every process p and sequence $s \in RAct^*$ such that $p \Downarrow s$ the set $\mathcal{A}(p,s)$ is a finite set of sets.*

Proof: Follows directly from Corollary 2.10 if we normalise the derivations from p. Suppose s uses k bound variables and let $y_1, \ldots y_k$ be k names not occurring in p. Then let s' be the sequence obtained from s by using these variables, in order, as bound variables. Then consider the computation tree from p where the nodes are labelled by equivalence classes of processes with respect to α-conversion and the branches are labelled by actions from s' or by i representing internal moves. This is a finite tree and the elements of $\mathcal{A}(p,s)$ correspond to $Subj(p')$ where $[p']$ labels a leaf. It follows that $\mathcal{A}(p,s)$ is finite. □

Lemma 3.6 *If $p \equiv_\alpha q$ and s is harmless then $\mathcal{A}(p,s) = \mathcal{A}(q,s)$*

Proof: It follows directly form Proposition 2.7. □

With these acceptance sets we can mimic the alternative preorder from [Hen88]. First acceptance sets are compared in the standard way by saying $\mathcal{A} \ll \mathcal{B}$ if for every $B \in \mathcal{B}$ there is some $A \in \mathcal{A}$ such that $A \subseteq B$. This is then lifted to processes by:

Definition 3.7 For any two processes p, q let $p \ll q$ is for every sequence $s \in RAct^*$ $p \Downarrow s$ implies

- $p \Downarrow s$

- for some harmless s' such that $s \equiv_\alpha s'$, (i. e. $bn(s') \cap (fn(p) \cup fn(q)) = \emptyset$), $\mathcal{A}(p, s') \ll \mathcal{A}(q, s')$

□

First we show that this preorder is preserved by α-conversion.

Proposition 3.8 *If $p \equiv_\alpha q$ then $p \ll q$.*

Proof: From Proposition 3.3 we know that that $p \equiv_\alpha q$ and $p \Uparrow s$ implies $q \Uparrow s$ and the result therefore follows by the preceding lemma. □

This means that we can be relatively liberal with respect to α-conversion and therefore in the following proofs we will not pay much attention to it. In particular when using the definition of $\ll$ we may choose any convenient s' provided it is α-equivalent to s. Also because of Lemma 3.4 it should be clear that we could have replaced the use of $\Downarrow s$ in the definition with $\downarrow s$ provided we restrict attention to harmless sequences.

As with strong bisimulation, $\sim$ in [MPW92a], $\lesssim$ is not preserved by substitutions. For example

$$\overline{x}v.nil \mid y(z).nil \;\simeq\; \overline{x}v.y(z).nil + y(z).\overline{x}v.nil$$

but they are no longer equivalent when x is substituted for y because then the left hand side may perform an internal move to *nil*. However it is preserved by injective substitutions:

Proposition 3.9 *If σ is injective on $fn(p, q)$ then $p \lesssim q$ implies $p\sigma \lesssim q\sigma$.*

Proof: Follows from Propositions 3.3 and 3.6.

Suppose $p\sigma \Downarrow s'$. We first show that $q\sigma \Downarrow s'$. Let s be a harmless sequence and σ' an injective substitution such that $p\sigma \equiv_\alpha p\sigma'$, $q\sigma \equiv_\alpha q\sigma'$ and $\sigma(s) \equiv_\alpha s'$. Then by Proposition 3.3, part (3), $p \Downarrow s$ and therefore $q \Downarrow s$. Again using the same proposition $q\sigma' \Downarrow \sigma'(s)$, i. e. $q\sigma \Downarrow \sigma(s)$.

Now suppose that $A \in \mathcal{A}(q\sigma, s'')$ where s'' is harmless and $s' \equiv_\alpha s''$. So $q\sigma \stackrel{s''}{\Longrightarrow} r$ where r is stable and $A = Subj(r)$. Again let s and σ' be such that s is harmless, σ' is injective, $p\sigma \equiv_\alpha p\sigma'$, $q\sigma \equiv_\alpha q\sigma'$ and $\sigma'(s) = s''$. By Proposition 2.7 $q \stackrel{s}{\Longrightarrow} q'$ for some q' such that $r \equiv_\alpha q'\sigma'[\underline{bn}(s) \mapsto \underline{bn}(s'')]$. In particular q' is stable and so $p \stackrel{s}{\Longrightarrow} p'$ for some stable p' such that $Subj(p') \subseteq Subj(q')$. For convenience let σ'' denote the substitution $\sigma'[\underline{bn}(s) \mapsto \underline{bn}(s'')]$ which is injective. Again by Proposition 2.7 it follows that $p\sigma' \stackrel{s''}{\Longrightarrow} r'$ such that $r' \equiv_\alpha p'\sigma''$ and since $p\sigma \equiv_\alpha p\sigma'$ $p\sigma \stackrel{s''}{\Longrightarrow} r''$ such that $r'' \equiv_\alpha p'\sigma''$. By Lemma 2.5 r'' is stable and $Subj(r'') \subseteq Subj(r)$ follows because $Subj(r'') = Subj(p'\sigma'') = \sigma''(Subj(p'))$ while $Subj(r) = Subj(q'\sigma'') = \sigma''(Subj(q'))$. □

The main result we wish to show is that the two relations $\ll$ and $\lesssim$ coincide on processes. The proof is very similar in style to the corresponding proof in [Hen88] but the details are considerably different. We first show

Proposition 3.10 *If* $p \ll q$ *then* $p \sqsubseteq\!\!\!\approx\ q$.

Proof: The proof has the same structure as that for the corresponding result in [Hen88], Lemma 4.4.13, although the details are more complicated because of the different forms of communication allowed in the π-calculus. Suppose $p \ll q$ and p *must* e. We show q *must* e by examining an arbitrary computation from $e \mid q$:

$$e \mid q = r_0 \rightarrowtail r_1 \rightarrowtail \ldots \rightarrowtail r_k \rightarrowtail \ldots \qquad (*)$$

and proving that there is some e_n such that $e_n \in Succ$. The proof depends on whether the computation $(*)$ is finite or infinite. As an example we consider only the finite case. So we may assume that r_k is stable for some k. Each r_i is of the form $(\underline{v}_i)r'_i$ where the individual restricted names in the sequence $\underline{v}_i$ arise because of the possible use of the Close rule from the operational semantics. Nevertheless by concentrating on the interaction between the two processes the computation $(*)$ may be unzipped into two derivations from e, q respectively, which use only actions from Act and which show their individual contributions:

$$e = e_0 \stackrel{\overline{a}_1}{\Longrightarrow} \ldots e_j \ldots \stackrel{\overline{a}_k}{\Longrightarrow} e_m$$

and

$$q = q_0 \stackrel{a_1}{\Longrightarrow} \ldots q_j \ldots \stackrel{a_k}{\Longrightarrow} q_m.$$

These are such that for each j there exists an i such that $r'_i = e_j \mid q_j$ and if $a_i = x(v), \overline{a}_i = \overline{x}(v)$ then $r_{i-1} \rightarrowtail r_i$ is inferred using an instance of the rule Close. If on the other hand it is inferred using an instance of the rule Com then if $a_i = \overline{x}y$ we can assume that $\overline{a}_i$ is xy. Let s denote the sequence $a_1 \ldots a_k$. Then because r_k is stable we can further assume that $Subj(q_m) \cap Subj(e_m) = \emptyset$. Let s' be a harmless sequence such that $s \equiv_\alpha s'$ and therefore $\overline{s} \equiv_\alpha \overline{s'}$. From Proposition 2.7 part (3), $q \stackrel{s'}{\Longrightarrow} q'$ such that $q' \equiv_\alpha q_m\sigma$ where σ is the renaming $\{\underline{bn}(s')/\underline{bn}(s)\}$. In particular, from Proposition 2.7 parts (1) and (2), $Subj(q') = Subj(q_m)\sigma$, where the latter denotes $\{\,\sigma(x) \mid x \in Subj(q_m)\,\}$. Similarly $e \stackrel{\overline{s'}}{\Longrightarrow} e'$ such that $Subj(e') = Subj(e_m)\sigma$. Because σ is injective on $n(e', q)$ it follows that $Subj(e') \cap Subj(q') = \emptyset$. There are two cases to consider.

1. $p \Downarrow s$.

Then $q \Downarrow s$ and so there is a p' such that $p \stackrel{s'}{\Longrightarrow} p'$, p' stable and $Subj(p') \subseteq Subj(q')$, i. e. $Subj(p') \cap Subj(e') = \emptyset$. Because s' is harmless the derivations $e \stackrel{\overline{s'}}{\Longrightarrow} e'$ and $p \stackrel{s'}{\Longrightarrow} p'$ can be zipped together to form a computation

$$e \mid p \rightarrowtail \ldots \rightarrowtail e' \mid p'.$$

Since p *must* e there exists some e'_j in this computation such that $e'_j \in Succ$. But each e'_i in this computation is such that $e'_i \equiv_\alpha e_i\{\underline{bn}(s'_i)/\underline{bn}(s_i)\}$ where s'_i and s_i are the corresponding initial subsequences of s' and s respectively. So $e'_j \in Succ$ and therefore the original computation is successful.

2.$p \Uparrow s$.

If we assume that no e_i is in *Success* then we immediately contradict the fact that p *must* e. For there must exist some subsequence of s, s_1 and a derivation $p \stackrel{s'_1}{\Longrightarrow} p'$ where $s_1 \equiv_\alpha s'_1$ and $p' \uparrow$. Then as in the previous case we can transform the derivation into

one involving a harmless subsequence and combine this with a corresponding derivation from e to obtain an unsuccessful computation from $e \mid p$.
When the computation $(*)$ is infinite the only possibility we have not touched on is when the unzipped derivations are infinite and p converges on all subsequences. Here we make use of Corollary 2.10 which states that the computation trees from p and q are finite branching, modulo α-conversion. The details of how this is used may be found in Lemma 4.4.13 from [Hen88]. □

To prove the converse we need to define two sets of special tests one of which tests for convergence and the other which is capable of testing for the contents of $\mathcal{A}(p,s)$. The crucial point is to be able to distinguish between outputing a free name on a channel and outputing an internal link. To achieve this we use the fact that if a process outputs a free name this name must belong to the free names of the process. We first examine convergence.

Let X be a finite set of names. For each s in Act let the test $c(s)_X$ be defined as follows:

1. $c(\varepsilon)_X = \overline{\omega}z.nil \oplus \overline{\omega}z.nil$ where z is any name. For convenience we use $1.w$ to denote this process and w to denote $\overline{\omega}z.nil$.
2. $c(xy.s)_X = 1.w + \overline{x}y.c(s)_{X\cup\{y\}}$
3. $c(\overline{x}y.s)_X = 1.w + x(z).\ if\, z = y\ then\, c(s)_X\ else\, \omega \quad$ where $z \notin fn(c(s)_X, y)$
4. $c(\overline{x}(y).s) = 1.w + x(z).\ if\, z \in X\ then\, \omega\ else\, c(s\{z/y\})_{X\cup\{z\}} \quad$ where $z \notin fn(s, X)$.

Here we use $z \in X$ as the obvious abbreviation for $z = x_1 \vee z = x_2 \ldots \vee z = x_n$ where X is the set $\{x_1, \ldots, x_n\}$. First we need a lemma about the effect of substitutions on these tests.

Lemma 3.11 $c(s)_X\sigma \equiv_\alpha c(s\sigma)_{X\sigma}$.

Proof: By induction on the length of s. □

We can now prove the expected property of these tests.

Proposition 3.12 *If* $fn(p) \subseteq X$ *then* $p \Downarrow s$ *if and only if* p *must* $c(s)_X$

Proof: The proof is by induction on s and we examine one case when it has the form $\overline{x}(y).s'$.

First suppose that $p \Downarrow s$. Then certainly $p \downarrow$ and we use induction on this to show that p *must* $c(s)_X$. Let us say that $q\sqrt{succ}$, for any term q, if every computation from q is successful. Note that by Proposition 2.6 this relation is preserved by α-conversion. So we must show that $p \mid c(s)_X\sqrt{succ}$. This amounts to showing that if $p \mid c(s)_X \rightarrowtail r$ then $r\sqrt{succ}$. The proof depends on why this move is made. If it is because of an internal move from $c(s)_X$ then by construction $r\sqrt{succ}$ while if it is an internal move from p we may use induction on $p \downarrow$. So we need only consider when it is because of a communication between the process and the test. There are two cases.

1. The rule Com is used in the derivation.
   Then r has the form $p' \mid \mathit{if}\, y \in X\ \mathit{then}\, \omega\ \mathit{else} \ldots$ where $p \xrightarrow{\overline{x}y} p'$. This means that $y \in fn(p) \subseteq X$ and therefore $r\surd\mathit{succ}$.

2. The rule Close is used in the derivation.
   Here r has the form

$$(y)(p' \mid (\ \mathit{if}\, y \in X\ \mathit{then}\, \omega\ \mathit{else}\, c(s'\{z/y\})_{X\cup\{z\}}\ )\{y/z\})$$

   for some new name z, where $p \xrightarrow{\overline{x}(y)} p'$. If $y \in X$ then this term is obviously in $\surd\mathit{succ}$. If not we know that $p' \Downarrow s'$ because $p \Downarrow s$ and since $fn(p') \subseteq X \cup \{y\}$ we may apply induction to obtain that $p' \mid c(s')_{X\cup\{y\}}\surd\mathit{succ}$. But by the previous lemma $c(s')_{X\cup\{y\}} \equiv_\alpha c(s'\{z/y\})_{X\cup\{z\}})\{y/z\}$ and therefore $c(s')_{X\cup\{y\}} \mid p' \equiv_\alpha c(s'\{z/y\})_{X\cup\{z\}})\{y/z\} \mid p'$. It now follows that $r\surd\mathit{succ}$.

Conversely suppose $p \mid c(s)_X\surd\mathit{succ}$. Obviously $p\downarrow$ and to show $p \Downarrow s$ it is sufficient to prove that if $p \stackrel{\overline{x}(y)}{\Longrightarrow} p'$ then $p \Downarrow s'$. Now it may not be possible for $c(s)_X$ to perform $\overline{x}(y)$ because y may be in X. So pick a completely new v. Then $p \stackrel{\overline{x}(v)}{\Longrightarrow} p'\{v/y\}$ and $p \mid c(s)_X \rightarrowtail r$ where up to α-conversion we may take r to be

$$(v)p'\{v/y\} \mid \mathit{if}\, v \in X\ \mathit{then}\, \omega\ \mathit{else}\, c(s'\{v/y\})_{X\cup\{v\}}\ .$$

Moreover we know that $r\surd\mathit{succ}$ and therefore that $p'\{v/y\} \mid c(s'\{v/y\})_{X\cup\{v\}}\surd\mathit{succ}$. By induction this means $p'\{v/y\} \Downarrow s'\{v/y\})$. The simple substitution $\{y/v\}$ is injective on $fn(p')$ and so we may apply Proposition 3.3 to conclude $p'\{v/y\}\{y/v\} \Downarrow s'\{v/y\}\{y/v\}$, i.e. $p' \Downarrow s'$. □

We next design a test $e(s,B)_X$, where $s \in RAct^*$ and B a finite subset of $\mathcal{N}$ with the property that whenever $p \Downarrow s$ and $fn(p) \subseteq X$

$$p\ \mathit{must}\, e(s,B)_X \iff \forall A \in \mathcal{A}(p,s)\ \ B \cap A \neq \emptyset.$$

Note that the right hand side is trivially satisfied if $\mathcal{A}(p,s) = \emptyset$. First let $e(x)$, $e(\overline{x})$ denote the tests $\overline{x}y.\overline{\omega}y.nil, x(y).\overline{\omega}y.nil$ respectively, for any name y. Then we define $e(s,B)_X$ by induction on s:

1. $e(\varepsilon,B)_X = \sum\{\, e(y) \mid y \in B\,\}$
2. $e(xy.s,B)_X = 1.w + \overline{x}y.e(s,B)_{X\cup\{y\}}$
3. $e(\overline{x}y.s,B)_X = 1.w + x(z).\ \mathit{if}\, z = y\ \mathit{then}\, c(s,B)_X\ \mathit{else}\, \omega$ where z is a new name
4. $e(\overline{x}(y).s,B)_X = 1.w + x(z).\ \mathit{if}\, z \in X\ \mathit{then}\, \omega\ \mathit{else}\, e(s\{z/y\},B)_{X\cup\{z\}}$ where z is a new name.

Proposition 3.13 *If $p \Downarrow s$ and $fn(p) \subseteq X$ then*

$$p\ \mathit{must}\, e(s,B)_X \iff \forall A \in \mathcal{A}(p,s)\ \ B \cap A \neq \emptyset.$$

Proof: The proof is by induction on s and again we examine only one case, when s has the form $\overline{x}(y).s'$

First suppose that $p \mid e(s,B)_X\surd succ$ and $A \in \mathcal{A}(p,s)$. We must show that $B \cap A \neq \emptyset$. We know that $p \stackrel{x(y)}{\Longrightarrow} p'' \stackrel{s'}{\Longrightarrow} p'$ for some stable p' such that $A = Subj(p')$. Because y may appear free in the test $e(s.B)_X$ we may not be able to use y in a communication between the process and the test. So choose a new v and by Proposition 2.7 we have, up to α-conversion, $p \stackrel{x(v)}{\Longrightarrow} p''\{v/y\} \stackrel{s\{v/y\}}{\Longrightarrow} p'\{v/y\}$. Moreover by Lemmas 2.4 and 2.5 and Proposition 2.6 it follows that $A\{v/y\} = Subj(p'\{v/y\})$. Because v is new we now have that, again up to α-conversion,

$$p \mid e(s,B)_X \rightarrowtail^* (v)p''\{v/y\} \mid \ \mathit{if}\ v \in X\ \mathit{then}\ \omega\ \mathit{else}\ e(s'\{v/y\}, B\{v/y\})_{X\cup\{v\}}\ .$$

Here we have used an analogue to Lemma 3.11 for the tests, namely that $(e(s,B)_X)\sigma \equiv_\alpha e(\sigma(s), B\sigma)_{X\sigma}$. From this it follows that $p''\{v/y\} \mid e(s'\{v/y\}, B\{v/y\})_{X\cup\{v\}})\surd succ$. So by induction $A' \cap B\{v/y\} \neq \emptyset$ for every $A' \in \mathcal{A}(p''\{v/y\}, s'\{v/y\})$. One such A' is $A\{v/y\}$ and so $A \cap B \neq \emptyset$, because v is new.

Conversely suppose that for all $A \in \mathcal{A}(p,s)$ $A \cap B \neq \emptyset$. We show that $p \mid e(s,B)_X\surd succ$. We know that $p \downarrow$ and the proof proceeds by induction on this fact. Suppose $p \mid e(s,B)_X \rightarrowtail r$. We must show that $r\surd succ$. If this move is because of an internal move of either the process or the test we can apply induction or else the result follows trivially by the construction of the tests. So we need only consider the case when there is communication between the process and the test. We consider the case when this is because of an application of the rule Close. The other possibility, when the rule Com is used, is left to the reader. Then r must have the form

$$(v)(p' \mid \ \mathit{if}\ v \in X\ \mathit{then}\ \omega\ \mathit{else}\ e(s'\{v/y\}, B\{v/y\})_{X\cup\{v\}}\ ),$$

up to α-conversion, where $p \stackrel{x(v)}{\longrightarrow} p'$. It is sufficient to consider the case when v is not in X when effectively any continuing computation is from $p' \mid e(s'\{v/y\}, B\{v/y\})_{X\cup\{v\}}$. So the result will follow by induction if we can show that for every $A' \in \mathcal{A}(p', s'\{v/y\})$ $A' \cap B\{v/y\} \neq \emptyset$. One can show that any such A' has the form $A\{v/y\}$ where $A \in \mathcal{A}(p,s)$. Since $A \cap B \neq \emptyset$ this implies $A' \cap B\{v/y\} \neq \emptyset$. □

With these two proposition we can now prove the converse of Proposition 3.10 and therefore the alternative characterisation of $\sqsubseteq$.

Theorem 3.14 *For every pair of processes p, q, $p \sqsubseteq q$ if and only if $p \ll q$.*

Proof: We need only prove $p \ll q$ implies $p \sqsubseteq q$ and this follows directly from the previous two propositions. For example suppose that $p \Downarrow s$, $q \Downarrow s$ and $B \in \mathcal{A}(q, s')$ where s' is new. We derive a contradiction from the assumption that for all $A \in \mathcal{A}(p, s')$ $A \not\subseteq B$. For each such A there must be some x_A in A and not in B. Let $L = \{\, a_A \mid A \in \mathcal{A}(p,s) \,\}$ and choose X so that it contains both $fn(p)$ and $fn(q)$. Then p *must* $e(s', L)_X$ whereas q need not always pass $e(s', L)_X$ and this contradicts the fact that $p \sqsubseteq q$. □

This theorem also shows that the behavioural preorder $\sqsubseteq\!\!\!\!\sim$ is determined by a small collection of tests, namely all those of the form $e(s,B)_X$ or $c(s)_X$. We call this set of tests *CTest* and they will be used in the next section.

As an application of the alternative characterisation we show that $\sqsubseteq\!\!\!\!\sim$ is preserved by most of the operators of the language.

Proposition 3.15 *For every operator op in* Σ $p_i \sqsubseteq\!\!\!\!\sim q_i$ *implies* $op(\ldots,p_i,\ldots) \sqsubseteq\!\!\!\!\sim op(\ldots,q_i,\ldots)$.

Proof: For the operator $|$ it is best to prove this directly from the definition of $\sqsubseteq\!\!\!\!\sim$ using the fact that $p \mid q$ *must* e if and only if p *must* $q \mid e$. For the other operators is is easier to prove the result for $\ll$. The only non-trivial case is for the binding operator $(y)-$. As an example of the proof technique let us show that if $p \ll q$ and $(y)q \Uparrow s$ then $(y)p \Uparrow s$. So without loss of generality we can suppose that $(y)q \stackrel{s}{\Longrightarrow} r$ where $r \uparrow$. If the rule Open is not used in this derivation then r has the form $(y)q'$ where $q \stackrel{s}{\Longrightarrow} q'$. So $q \Uparrow s$ from which it follows that $p \Uparrow s$ and therefore $(y)p \Uparrow s$ since y can not appear in s. So suppose Open is used. Then the derivation can be viewed as

$$(y)q \stackrel{s_1}{\Longrightarrow} (y)q_1 \stackrel{\overline{x}(v)}{\longrightarrow} q_1'\{v/y\} \stackrel{s_2}{\Longrightarrow} r$$

where

$$q \stackrel{s_1}{\Longrightarrow} q_1 \stackrel{\overline{x}y}{\longrightarrow} q_1'$$

and $v \notin fn((y)q_1')$. This means that the substitution $\{v/y\}$ is injective on $fn(q_1')$ and therefore we can apply Proposition 2.7 to find a q' and s_2' such that

$$q_1' \stackrel{s_2'}{\Longrightarrow} q', \quad s_2 \equiv_\alpha s_2' \text{ and } r \equiv_\alpha q'\{v/y\}\{\underline{bn}(s_2)/\underline{bn}(s_2')\}.$$

If we take care to arrange so that $\underline{bn}(s_2)$ are all new names, which is possible using α-conversion, then the renaming is injective and using Proposition 3.3 we have that $q' \uparrow$. So $q \Uparrow s$ and therefore $p \Uparrow s$. From this it is easy to establish that $(y)p \Uparrow s$. □

As with the semantic equivalence of [MPW92a], $\sqsubseteq\!\!\!\!\sim$ is not preserved by prefixing by input actions; for example if x and y are different names then

$$\mathit{if}\ x = y\ \mathit{then}\ \overline{v}y.nil\ \mathit{else}\ \overline{v'}x.nil \quad \sqsubseteq\!\!\!\!\sim\ \overline{v'}x.nil$$

but this is not the case if both sides are prefixed by $z(x)$. However the following proposition gives us a finitary rule for inferring $z(x).p \sqsubseteq\!\!\!\!\sim z(x).q$ — which is the same as that used in [MPW92a].

Proposition 3.16 *If for all* $v \in fn(p,q,x)$ $p\{v/x\} \sqsubseteq\!\!\!\!\sim q\{v/x\}$ *then* $z(x).p \sqsubseteq\!\!\!\!\sim z(x).q$.

Proof: We use the alternative characterisation of $\sqsubseteq\!\!\!\!\sim$, the relation $\ll$. Every sequence from $z(x).p$ or $z(x).q$, apart from the empty one, is of the form $zv.s$ for some v, where s is a sequence from $p\{v/x\}$ or $q\{v/x\}$ respectively. If $v \in fn(p,q,x)$ then we can use $p\{v/x\} \ll q\{v/x\}$ to compare their acceptance sets and convergence. Otherwise $\{v/x\}$ is injective on $fn(p,q)$ and therefore by Proposition 3.9 $p\{v/x\} \ll q\{v/x\}$ and once more we can carry out the comparisons. □

4 Modelling the Language L_π

In this section we address the question of finding a denotational semantics for the language L_π. We first discuss what should be an appropriate class of interpretations and then construct a *fully-abstract* one in the sense that it identifies and only identifies processes which are testing equivalent. This is a term model which is constructed using an equationally based proof system and therefore we have as a by-product a sound and complete proof system for testing equivalence over processes. We also show that the particular interpretation we construct is initial in the class of all fully-abstract interpretations.

Because of the computational nature of the language we expect to interpret it in some *complete partial order* or *cpo* D where recursive definitions can be interpreted. We can follow the standard paradigm of *algebraic semantics*, [Gue81], [Hen88], if in addition we associate a continuous function with each of the operators of the language. For the most part the types of these operators are straightforward. Recall that Σ represents the set of operators

$$\{\ nil,\ \Omega,\ (y),\ \overline{x}y,\ +,\ \oplus,\ |\ \}.$$

Each of these expects to be applied to either zero, one or two processes and returns a process. So they may be interpreted as continuous functions over D of the appropriate arity. However the input operator is more subtle; it is a binding operator in that in $x(y).t$ all free occurrences of the name y in t are bound by the prefix $x(y)$. So an appropriate type for an input function *in* is

$$\mathcal{N} \times (\mathcal{N} \longmapsto D) \longmapsto D.$$

Then $x(y).t$ will be $in(x, f)$ where f represents the function $\lambda y.[\![t]\!]$. This leads to the definition of what we call a *natural interpretation* for the language L_π.

Definition 4.1 A *natural interpretation* (for the language L_π), consists of $\langle D, in_D \rangle$ where

i) D is a $\Sigma - cpo$

ii) $in_D : \mathcal{N} \times (\mathcal{N} \longmapsto D) \longmapsto D$ is a function continuous in its second argument, where $\mathcal{N} \longmapsto D$ inherits the natural pointwise ordering from D

□

This extra structure brings us outside the standard framework of algebraic semantics as for example defined in [GTWW77], [Gue81]. but homomorphisms may be defined as the obvious generalisation of homomorphisms for $\Sigma - cpos$:

Definition 4.2 A *homomorphism* from the natural interpretation $\langle C, in_C \rangle$ to the natural interpretation $\langle D, in_D \rangle$ is a Σ-homomorphism h of the underlying Σ-cpos which satisfies in addition

$$h(in_C(x, f)) = in_D(x, h \cdot f).$$

□

Given such a natural interpretation, D, we can define a semantic interpretation of L_π following the usual approach of denotational semantics. We let Env_D be the set of D-environments, i.e. mappings from PV to D, ranged over by ρ and we assume an evaluation function $[\![\ ]\!]: BExp \longmapsto \{tt, ff\}$. Then the semantics of the language L_π is given as a function:

$$D[\![\ ]\!]: L_\pi \longmapsto (Env_D \longmapsto D)$$

and is defined by structural induction:

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

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) $D[\![\ if\ be\ then\ t\ else\ u\ \ ]\!]\rho\ =\ \begin{array}{l} D[\![t]\!]\rho \ \text{ if } \ [\![be]\!] = tt \\ D[\![u]\!]\rho \ \text{ if } \ [\![be]\!] = ff \end{array}$

v) $D[\![x(y).t]\!]\rho = in_D(x, \lambda y.D[\![t]\!]\rho)$

where Y is the least-fixpoint operator for continuous functions in D. Although we are somewhat outside the realm of standard algebraic semantics, [Gue81], many of the usual techniques and results apply. One particular property we will use is that the denotation of a process is completely determined by that of its finite approximations.

Definition 4.3 Let $\preceq$ be the least reflexive, transitive relation over L_π which is preserved by all the operators in $\Sigma \cup \{x(y)\}$ and which satisfies

1. $\Omega \preceq t$
2. $t\{rec\,X.\ t/P\} \preceq rec\,X.\ t$

□

We use FL_π to denote the set of syntactically finite terms, i.e. all terms in L_π which have no occurrence of the recursion construct, $rec\,X.\ -$. Then we define the set of *finite approximations* to a term t as $\mathrm{App}(t) = \{\, d \in FL_\pi \mid d \preceq t \,\}$. We state without proof:

Theorem 4.4 *For every interpretation D $D[\![t]\!] = \bigvee\{\, D[\![d]\!] \mid d \in App(t) \,\}$.*

Of course there are many interpretations which are of no interest whatsoever. However we show that there is at least one which reflects the behavioural view of processes as outlined in the previous section.

Definition 4.5 An interpretation D is a *model* for the language L_π if for every pair of processes p, q, $D[\![p]\!] \leq D[\![q]\!]$ implies $p \sqsubseteq q$. It is a *fully-abstract* model if in addition $p \sqsubseteq q$ implies $D[\![p]\!] \leq D[\![q]\!]$. □

$$
\begin{array}{rcl}
X \oplus (Y \oplus Z) & = & (X \oplus Y) \oplus Z \\
X \oplus Y & = & Y \oplus X \\
X \oplus X & = & X \\
X + (Y + Z) & = & (X + Y) + Z \\
X + Y & = & Y + X \\
X + X & = & X \\
X + nil & = & X \\
pre.X + pre.Y & = & pre.(X \oplus Y) \\
x(y).X + x(y).Y & = & x(y).X \oplus x(y).Y \\
\overline{x}y.X + \overline{x}y'.Y & = & \overline{x}y.X \oplus \overline{x}y'.Y \\
X + (Y \oplus Z) & = & (X + Y) \oplus (X + Z) \\
X \oplus (Y + Z) & = & (X \oplus Y) + (X \oplus Z) \\
X \oplus Y & \leq & X \\
X + \Omega & \leq & \Omega \\
\Omega & \leq & X
\end{array}
$$

$$
\begin{array}{rcl}
(x)(X + Y) & = & (x)X + (x)Y \\
(x)(X \oplus Y) & = & (x)X \oplus (x)Y \\
(x)(pre.X) & = & pre.(x)X \quad \text{if } x \text{ not in } n(pre) \\
(x)(pre.X) & = & nil \quad \text{if } x \text{ is the subject of } pre \\
(x)X & = & X \quad \text{if } x \notin fv(X) \\
(x)(y)X & = & (y)(x)X \\
(x)\Omega & = & \Omega
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{if}\ tt\ \mathit{then}\ X\ \mathit{else}\ Y & = & X \\
\mathit{if}\ ff\ \mathit{then}\ X\ \mathit{else}\ Y & = & Y
\end{array}
$$

Figure 3: Basic Equations

We construct a fully-abstract model using a proof system based on a set of inequations.

The inequations are given in Figures 3 and 4. The first contain the standard equations for testing from [Hen88] and the additional one,

$$\overline{x}y.X + \overline{x}y'.Y = \overline{x}y.X \oplus \overline{x}y'.Y,$$

from [HI91] which is appropriate to value passing process calculi. It also contains natural laws for the restriction operator, taken from [MPW92a] and obvious rules for the *if* ... *then* ... construct. In the second we have rules governing the parallel combinator,

the principal being a version of the interleaving law. Unlike the standard theories of concurrency, such as that in [Mil89], the restriction operator can not be eliminated from all finite terms using the equations; this is a reflection of the extra power of restriction in the π-calculus. However the irreducible occurrences can be coded up as a form of derived prefix.

Definition 4.6 If $x \neq y$ then $\overline{x}(y)p$ is a shorthand for the term $(y)\overline{x}y.p$ and the subject of the prefix $\overline{x}(y)$ is $\overline{x}$. □

This notation, taken from [MPW92a], is used in the interleaving law where it is assumed that *pre* now ranges over both the standard prefixes of the form $x(y), \overline{x}y$ and the new derived prefix $\overline{x}(y)$. The proof system itself is given in Figure 5 and is a simple extension of equational reasoning. The principle new rule is the Input rule for deriving statements of the form $x(y).p \leq x(y).q$. The rule which one might suggest for input prefixing,

$$\frac{p \leq q}{x(y).p \leq x(y).q}$$

is not sound because $\sqsubseteq_{\approx}$ is not preserved by input prefixing. For another example let p, q be $z(v).\overline{y}w.nil + \overline{y}w.z(v).nil$, $z(v).nil \mid \overline{y}w.nil$, respectively; then $p \sqsubseteq_{\approx} q$ but $x(y).p \not\sqsubseteq_{\approx} x(y).q$ because $x(y).q$ can perform the action $x(z)$ followed by a communication and be transformed into *nil* whereas $x(y).p$ does not have that possibility. So $x(y).p$ must satisfy the test $\overline{x}z.(z(v').\overline{\omega} + \overline{z}(v').\overline{\omega}$ but $x(y).q$ may fail it. For this reason we need the more complicated Input rule.
We write $\vdash t \leq u$ if $t \leq u$ can be derived in this proof system.

Proposition 4.7 *(Soundness) If* $\vdash t \leq u$ *then* $t \sqsubseteq_{\approx} u$.

Proof: It is sufficient to check that all of the equations are satisfied by $\sqsubseteq_{\approx}$ and that it is preserved by the rules. The soundness of Rule Input follows from Proposition 3.16 while that of Rule α follows from Proposition 3.8. □

Note that Proposition 3.16 would justify a more restricted form of the input rule, namely

$$\frac{p\{z/y\} \leq q\{z/y\} \text{ for all free names } z \in fv(p, q, y)}{x(y).p \leq x(y).q}$$

However in later proofs it is more convenient to have the more general rule in the proof system.

The proof system is also complete for finite processes. These are closed terms from FL_{π}, the sublanguage which does not use the recursion construct. However we can prove a slightly more general result. Let us add to the proof system the following rule for recursive equations, which we call the Unwind rule:

$$\frac{u \leq t\{rec\, X.\ t/X\}}{u \leq rec\, X.\ t}$$

Then we can show that the augmented proof system, whose provability relation we denote by $\vdash_r$, is complete for statements of the form $d \sqsubseteq_{\approx} p$ where d is a finite closed term and p is any closed term. The essential idea behind the proof is *head normal forms*.

$$
\begin{array}{rcl}
(X \oplus Y) \mid Z & = & X \mid Z \oplus Y \mid Z \\
X \mid (Y \oplus Z) & = & X \mid Y \oplus X \mid Z \\
nil \mid P & = & P \mid nil \; = \; P \\
X \mid (Y + \Omega) & = & (X + \Omega) \mid Y \; = \; \Omega
\end{array}
$$

Let X, Y denote $\sum\{pre_i.X_i, i \in I\}$, $\sum\{pre_j.Y_j, j \in J\}$, where no pre_i (resp. pre_j) binds a name free in Y (resp. X_i). 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

$$
\begin{array}{l}
ext(X,Y) = \sum\{pre_i.(X_i \mid Y),\ i \in I\} + \sum\{pre_j.(X \mid Y_j),\ j \in J\} \\
int(X,Y) = \sum\{Z_{i,j},\ pre_i \ \ comp \ \ pre_j\}
\end{array}
$$

where $comms(X,Y)$ is true if there is no i,j such that $pre_i \ \ comp \ \ pre_j$ and $pre_i \ \ comp \ \ pre_j$ holds when

1. pre_i is $\overline{x}u$ and pre_j is $x(v)$; then $Z_{i,j}$ is $X_i \mid Y_j\{u/v\}$
2. pre_i is $\overline{x}(u)$ and pre_j is $x(v)$; then $Z_{i,j}$ is $(w)(X_i\{w/u\} \mid Y_j\{w/v\})$ where w is not free in $(u)X_i$ or in $(v)Y_j$
3. pre_i is $x(v)$ and pre_j is $\overline{x}u$: then $Z_{i,j}$ is $X_i\{u/w\} \mid Y_j$
4. pre_i is $x(v)$ and pre_j is $\overline{x}(u)$; then $Z_{i,j}$ is $(w)(X_i\{w/v\} \mid Y_j\{w/u\})$ where w is not free in $(v)X_i$ or in $(u)Y_j$

Figure 4: Equations for $\mid$

Definition 4.8 Let $\mathcal{A}$ be a finite collection of finite subsets of $\mathcal{N} \cup \overline{\mathcal{N}}$, and for each $a \in \cup\mathcal{A}$ let p_a be a term given as follows:

1. If $a \in \mathcal{N}$ then p_a is $a(y).p'$ for some term p'
2. If $a \in \overline{\mathcal{N}}$ then p_a is a non-empty sum of the form $ay_1.p_1 + \ldots + ay_n.p_n + \{\ a(y).p'\}$ where p_i, p' are arbitrary terms and y_i are distinct names and $+\{\ \}$ indicates an optional summand.

Then

$$
\sum\{\sum\{p_a \mid a \in A\} | A \in \mathcal{A}\}
$$

is a *hnf*. □

Note that by taking $\mathcal{A}$ to be empty we have that *nil* is a hnf. We first show that every convergent process can be reduced to a hnf which has the same "size" for some measure of size. Here we take the size of a a process p, $\mid p \mid$, to be the length of the longest sequence of actions which it can perform.

Lemma 4.9 *If $p \downarrow$ then there exists a hnf, hnf (p), such that $\vdash_r p = hnf\,(p)$ and $\mid p \mid = \mid hnf(p) \mid$.*

Proof: It is virtually identical to that of Proposition 4.2.1 of [HI91] and is therefore omitted. The only new ingredient is that in the subterms $p_{\overline{x}}$ there is at most one summand of the form $\overline{x}(y).p'$. If during the reduction procedure two such summands are generated then they can be replaced by one using the equations as follows:

$$\begin{aligned}
\overline{x}(y).p' + \overline{x}(z).p'' &= \overline{x}.(w)p'\{w/y\} + \overline{x}.(w)p''\{w/z\} \quad \text{by } \alpha\text{-conversion, where } w \text{ is new} \\
&= (w)(\overline{x}w.p'\{w/y\} + \overline{x}w.p''\{w/z\}) \\
&= (w)(\overline{x}w.(p'\{w/y\} \oplus p''\{w/z\})) \\
&\equiv \overline{x}(w).(p'\{w/y\} \oplus p''\{w/z\})
\end{aligned}$$

Note also that the new operator, (y), is accommodated by the restriction axioms in Figure 3. If h is a hnf then these can be used to reduce $(y)h$ to a hnf . □

Theorem 4.10 *(Partial Completeness) For all finite processes d and arbitrary processes p, $d \sqsubseteq\!\!\!\sim\ p$ implies $\vdash_r d \leq p$.*

Proof: (Outline) The proof is by induction on $\mid d \mid$. If $d \uparrow$ then it is easy to show that $\vdash d = \Omega$ and therefore the result follows trivially. So we may assume both d and p converge and therefore by the previous result that they are head normal forms. The proof is now virtually identical to that of Theorem 5.6 of [HI91]. □

The term model is constructed from this provability relation applied to what we call *behaviourally finite* terms. Intuitively these are terms which can only ever perform a finite number of possible actions. But because of the nature of bound output actions the definition is a little complicated. First let $(a, p) \equiv (a', p')$ if

1. $a = x(y)$ or $\overline{x}y$ implies $a' = a$ and $p' = p$
2. $a = \overline{x}(y)$ implies $a' = \overline{x}(z)$ and $p' \equiv_\alpha p\{z/y\}$ for some z not free in $(y)p$.

It is easy to check that $\equiv$ is in fact an equivalence relation.

Definition 4.11 The set of behaviourally finite terms, *BF*, is the least set of (syntactically) finite terms b such that

1. the set $\{\,[(a, b'])\mid b \stackrel{a}{\Longrightarrow} b' \text{ and } b' \not\sqsubseteq\!\!\!\sim\ \Omega\,\}$ is finite
2. whenever $b \stackrel{a}{\Longrightarrow} b'$ for any action a then b' is also in *BF*.

□

I $\quad \dfrac{}{p \leq p} \qquad \dfrac{p \leq q,\ q \leq r}{p \leq r}$

II $\quad \dfrac{p_i \leq q_i, \quad 1 \leq i \leq n}{op(p_1, \ldots, p_n) \leq op(q_1, \ldots, q_n)}$ for every $op \in \{\overline{x}y., (x), +, \oplus, |\}$

Eq $\quad \dfrac{}{p \leq q}$ for every instance of an inequation

α $\quad \dfrac{p \equiv_\alpha q}{p \leq q}$

Input $\quad \dfrac{p\{z/y\} \leq q\{z/y\} \ \text{ for all names } \ z}{x(y).p \leq x(y).q}$

If1 $\quad \dfrac{p \leq p'}{\mathit{if}\ be\ \mathit{then}\ p\ \mathit{else}\ q \ \leq \ \mathit{if}\ be\ \mathit{then}\ p'\ \mathit{else}\ q}$

$\dfrac{q \leq q'}{\mathit{if}\ be\ \mathit{then}\ p\ \mathit{else}\ q \ \leq \ \mathit{if}\ be\ \mathit{then}\ p\ \mathit{else}\ q'}$

If2 $\quad \dfrac{}{\mathit{if}\ tt\ \mathit{then}\ p\ \mathit{else}\ q \ = p}$

$\dfrac{}{\mathit{if}\ ff\ \mathit{then}\ p\ \mathit{else}\ q \ = q}$

Figure 5: The Proof System

A typical example of a finite process which is not behaviourally finite is $x(y).nil$ because it can perform an infinite number of distinct actions. In any model of the language this term will not be interpreted as a compact element. One can also check that the prefix operator $x(y)$ does not preserve the property of being behaviourally finite. As a simple example nil is in BF but $x(y).nil$ is not. However it is preserved by all the other operators and therefore when constructing the model we will only have to pay particular attention to this operator, $x(y)$.

For convenience let us use $p \leq_E q$ in place of $\vdash p \leq q$ and when appropriate $p \leq_{Er} q$ in place of $\vdash_r p \leq q$. The relation $\leq_E$ is a preorder over the set BF and is also preserved by all the operators in Σ. So let P_E denote the Σ-po (or Σ-partial order) whose carrier is the set of equivalence classes $[b]$ over BF, generated by the kernel of $\leq_E$. These are ordered by $[d] \leq [d']$ if $d \leq_E d'$ and the operations are defined by $op_{P_E}(\ldots, [d], \ldots) = [op(\ldots, d, \ldots)]$. This is extended to a Σ-cpo in the standard way by ideal completion. We denote the resulting *cpo* by C_E. Briefly speaking the elements of C_E may be taken to be *ideals* over P_E, i. e. non-empty downward-closed subsets of P_E ordered by set inclusion. The

operations are defined pointwise:

$$op(\ldots, I, \ldots) = \{\, op_{P_E}(\ldots, e, \ldots) \mid e \in I \,\}\downarrow$$

where $S\downarrow$ denotes the downward closure of the set S. These definitions ensure that C_E is a Σ-cpo. More details of this construction, and its properties, may be found in [Gue81], [Hen88]. To consider it as an interpretation it is sufficient to define in_{C_E} to interpret the missing operator, $x(y)$. Since $\mathcal{N}$ is countable we can assume that there exists a sequence of finite subsets of $\mathcal{N}$, $\mathcal{N}^k, k \geq 0$, such that $\mathcal{N} = \cup_k \mathcal{N}^k$. These finite sets are used to define approximations to in_{C_E}: for each $k \geq 0$ let

$$in^k_{C_E} : \mathcal{N} \times (\mathcal{N} \longmapsto C_E) \longmapsto C_E$$

be defined by

$$in^k_{C_E}(x, f) = \{\, [x(n).\ if\ n \in \mathcal{N}^k\ then\ b\ else\ \Omega\ ] \mid b \in f(n) \,\}\downarrow .$$

Here we are assuming that the language for boolean expressions is sufficiently powerful to express "$n \in \mathcal{N}^k$". Since $\mathcal{N}^k$ is finite this is not unreasonable. We may now define

$$in_{C_E}(x, f) = \bigvee \{\, in^k_{C_E}(x, f) \mid k \geq 0 \,\}.$$

It is easy to check that each $in^k_{C_E}$ is continuous and therefore so is in_{C_E}. Thus we can consider C_E to be an interpretation and so we can interpret our language in it. We show that it gives a fully-abstract model.

The principle characteristic of this interpretation is given in the following theorem.

Theorem 4.12 *For every $b \in BF$ $C_E[\![b]\!] = \{[b]\}\downarrow$.*

Proof: Since all elements of BF are finite terms we can prove the result by induction on the size of the terms, i. e. the number of symbols it contains. The only non-trivial case is when the term has the form $x(y).b$. By induction we may assume $C_E[\![b\{z/y\}]\!] = \{[b\{z/y\}]\}\downarrow$ for every name z. So by definition

$$C_E[\![x(y).b]\!] = \bigvee_k in^k_{C_E}(x, \lambda n.\{[b\{n/y\}]\}\downarrow)$$

Using the inductive hypothesis one can check that for each k

$$in^k_{C_E}(x, \lambda n.\{[b\{n/y\}]\}\downarrow) = \{[x(n).\ if\ n \in \mathcal{N}^k\ then\ b\{n/y\}\ else\ \Omega\ ]\}\downarrow .$$

Let m be such that $x \notin \mathcal{N}^m$ implies $b\{x/y\} \precsim \Omega$ and therefore $\vdash b\{x/y\} \leq \Omega$. Then using the Input rule it follows that for every k

$$\vdash x(n).\ if\ n \in \mathcal{N}^k\ then\ b\{n/y\}\ else\ \Omega\ = x(n).\ if\ n \in \mathcal{N}^m\ then\ b\{n/y\}\ else\ \Omega$$

and therefore

$$C_E[\![x(y).b]\!] = \{[x(n).\ if\ n \in \mathcal{N}^m\ then\ b\{n/y\}\ else\ \Omega\ ]\}\downarrow .$$

An application of the Input rule also gives

$$\vdash x(n).\ if\ n \in \mathcal{N}^m\ then\ b\{n/y\}\ else\ \Omega\ = x(y).b$$

and it follows that $C_E[\![x(y).b]\!] = \{[x(y).b]\}\downarrow$. □

We now wish to show that the provability relation $\vdash_r$ is sound and complete with respect to the model C_E for statements of the form $b \leq q$. To this end we extend the syntax of the language by allowing, for each $k \geq 0$, the input prefix $x^k(y)$. The semantics of this new construct is obvious:

$$C_E[\![x^k(y).t]\!]\rho = in^k_{C_E}(x, \lambda y.C_E[\![t]\!]\rho).$$

This extended syntax allows us to define syntactically the behaviourally finite terms which determine the semantics of arbitrary finite terms.
For any finite process d let $d^{(m)}$ be defined by

1. $d^{(0)} = \Omega$
2. $(x(y).d)^m = x^m(y).d^{(m)}$
3. $(op(\ldots, d, \ldots))^{(m)} = op(\ldots, d^{(m)}, \ldots)$.

Note that each $d^{(m)}$ is behaviourally finite and one can show that $n \leq m$ implies $\vdash d^{(n)} \leq d^{(m)}$ and that $\vdash d^{(m)} \leq d$ for any m. These proofs make essential use of the Input rule. But the most important property of these behaviourally finite approximations is:

Proposition 4.13 *For all finite d, $C_E[\![d]\!] = \bigvee_m C_E[\![d^{(m)}]\!]$.*

Proof: By structural induction on d. The only nontrivial case is when d has the form $x(y).e$.

$$\begin{aligned}
C_E[\![x(y).e]\!] &= \bigvee_m in^m_{C_E}(x, \lambda w.C_E[\![e\{w/y\}]\!]) \\
&= \bigvee_m in^m_{C_E}(x, \lambda w.\bigvee_k C_E[\![e^{(k)}\{w/y\}]\!]) \quad \text{by induction} \\
&= \bigvee_k in^m_{C_E}(x, \lambda w.C_E[\![e^{(m)}\{w/y\}]\!]) \quad \text{by continuity} \\
&= \bigvee_m [\![x^m(y).e^{(m)}]\!].
\end{aligned}$$

□

As an immediate corollary we have a partial completeness result:

Proposition 4.14 *For every $b \in BF$ and process q $C_E[\![b]\!] \leq C_E[\![q]\!]$ implies $\vdash_r b \leq q$.*

Proof: First recall that $C_E[\![q]\!] = \bigvee\{ C_E[\![d]\!] \mid d \preceq q \}$ and that $C_E[\![b]\!]$ is a compact element since, by Theorem 4.12, it has the form $\{[b]\}\downarrow$. So for some $d \preceq q$ $C_E[\![b]\!] \leq C_E[\![d]\!]$.
Also $C_E[\![d]\!] = \bigvee\{ C_E[\![d^{(m)}]\!] \mid m \geq 0 \}$ and again by the compactness of $C_E[\![b]\!]$ there is a $m \geq 0$ such that $C_E[\![b]\!] \leq C_E[\![d^{(m)}]\!]$. Both of these are in BF and so using Theorem 4.12 $\vdash b \leq d^{(m)}$.
Therefore $\vdash_r b \leq d^{(m)} \leq d \leq q$. □

We also have the converse:

Proposition 4.15 *For every $b \in BF$ and process q $\vdash_r b \leq q$ implies $C_E[\![b]\!] \leq C_E[\![q]\!]$.*

Proof: The proof follows from the property

$\vdash_r b \leq q$ implies that there exists a $d \preceq q$ such that
for some $m \geq 0$ $\vdash b \leq d^{(m)}$. $(*)$

This is proved by induction on the length of the proof of $\vdash_r b \leq q$ and proceeds by considering the last rule applied in the proof. There are only three non-trivial cases, the Input rule, the Unwind rule and the Transitivity rule. As an example we look at the Input rule. Here b, q have the form $x(y).b', x(y).p$ respectively and $\vdash_r b \leq q$ has been inferred because for all $z \in \mathcal{N}$ $\vdash_r b\{z/y\} \leq q\{z/y\}$. Moreover by induction the proposition will apply to all of these statements. Let $\mathcal{N}^n$ be the finite set $\{ z \mid \not\vdash b\{z/y\} \leq \Omega \}$. For each $z \in \mathcal{N}^n$ there is a d_z and n_z such that $\vdash b\{z/y\} \leq (d_z\{z/y\})^{(n_z)}$. Let $d \in \mathrm{App}(q)$ dominate all d_z, i. e. $d_z \preceq d$, and let n be the maximum of $\{n_z\}$. Then for every name z

$$\vdash b\{z/y\} \leq d\{z/y\}^n = d^n\{z/y\}.$$

Applying the Input rule we now obtain $\vdash x(y).b \leq x^n(y).d^n$

Assuming $(*)$ the proof of the proposition is now immediate. For if $\vdash_r b \leq q$ then let d be such that $d \preceq q$ and $\vdash b \leq d^{(m)}$ for some m. Since both of these are in BF it follows from Theorem 4.12 that $C_E[\![b]\!] \leq C_E[\![d^{(m)}]\!]$. Applying Theorem 4.4 and Proposition 4.13 we obtain $C_E[\![b]\!] \leq C_E[\![q]\!]$. □

We can now show that that the interpretation C_E is fully abstract with respect to a finitary version of the behavioural preorder.

Definition 4.16 If R is any relation over processes let $pR^f q$ if for every $b \in BF$ bRp implies bRq. □

Theorem 4.17 *C_E is a fully abstract model with respect to $\sqsubseteq_{\approx}^{f}$.*

Proof: For convenience let us denote $C_E[\![p]\!] \leq C_E[\![q]\!]$ by $p \leq_{C_E} q$. Since C_E is an algebraic cpo it follows that the relation $\leq_{C_E}$ is finitary, i. e. $\leq_{C_E} = \leq_{C_E}^f$. So

$$\begin{aligned}
p \leq_{C_E} q &\iff p \leq_{C_E}^f q \\
&\iff b \leq_{C_E} p \ \textit{ implies } \ b \leq_{C_E} q \\
&\iff \vdash_r b \leq p \ \textit{ implies } \ \vdash_r b \leq q \\
&\iff b \sqsubseteq_{\approx} p \ \textit{ implies } \ b \sqsubseteq_{\approx} q \\
&\iff p \sqsubseteq_{\approx}^{f} q
\end{aligned}$$

□

The required full-abstractness result will follow if we can show that $\sqsubseteq_{\approx}$ coincides $\sqsubseteq_{\approx}^{f}$. If we examine the tests which characterise $\sqsubseteq_{\approx}$ one can see why intuitively why this should be true; to guarantee each specific test in *CTest* only a finitary amount of the behaviour of a process is required. This could be proved directly using the operational semantics but we take advantage of the development of head normal forms in order to give a simpler proof.

Lemma 4.18 *For every test $e \in CTest$ p must e implies that there exists some $b \in BF$ such that $\vdash_r b \leq p$ and b must e.*

Proof: The proof proceeds by induction on the size of the test e. Because of the form of the tests in *CTest* p must converge and therefore it must have a hnf, say of the form

$$\sum \{\sum\{p_a \mid a \in A\} | A \in \mathcal{A}\}.$$

The idea is to replace each p_a with a behaviourally finite approximation which is provably less than it so that the resulting process still satisfies the test. The exact from the replacement term takes depends on the test e. We examine only one example. Suppose it is $e(xy.s, B)_X$, i. e. $1.\omega + \overline{x}y.e(s, B)_{X\cup\{y\}}$. Then each p_a other than one of the form $x(z).p'$, if it exists, is replaced by a term involving Ω. For example if it has the form $x'(z).p'$ where x' is different from x then it is replaced by $x'(z).\Omega$ and if it has the form $\overline{x'}y_1.q_1 + \ldots + \overline{x'}y_k.q_k \ \{\ + \overline{x'}(y).q'\}$ it is replaced by $\overline{x'}y_1.\Omega + \ldots + \overline{x'}y_k.\Omega \ \{\ + \overline{x'}(y).\Omega\}$. If one of the form $x(z).p'$ exists then $p'\{y/z\}$ *must* $e(s, B)_{X\cup\{y\}}$. So by induction there is a b' such that $\vdash_r b' \leq p'\{y/z\}$. So replace the $x(z).p'$ by $x(z).$ *if* $z = y$ *then* b' *else* Ω . Suppose that the result of all these replacements is b. Then b *must* $e(xy.s, B)_X$ and one can easily show that $\vdash_r b \leq p$. □

Corollary 4.19 *The relation $\sqsubseteq_{\approx}$ is finitary, i. e. $\sqsubseteq_{\approx} = \sqsubseteq_{\approx}^{f}$.*

Proof: It is sufficient to prove that $p \sqsubseteq_{\approx}^{f} q$ implies $p \sqsubseteq_{\approx} q$.
Let p *must* e where $e \in CTest$. Then there exists a $b \in BF$ such that $\vdash_r b \leq p$ and b *must* e. From the soundness of the proof system it follows that $b \sqsubseteq_{\approx} p$ and so $b \sqsubseteq_{\approx} q$; therefore q *must* e. □

We can now state the main result of the section

Theorem 4.20 *C_E is a fully abstract model with respect to $\sqsubseteq_{\approx}$.* □

Finally we can show that this particular fully abstract model is in some sense the "least" interpretation which is fully-abstract.

Definition 4.21 An interpretation C is *initial* in a class of models $\mathcal{C}$ if for every $D \in \mathcal{C}$ there is a unique homomorophism $i_D : C \longmapsto D$. □

Theorem 4.22 *C_E is initial in the class of fully-abstract models.*

Proof: Let D be a fully-abstract model. Define $i_D : C_E \longmapsto D$ by

$$i_D(I) = \bigvee \{\, D[\![b]\!] \mid [b] \in I \,\}.$$

Since D is fully-abstract this is well-defined and obviously is a Σ-homomorphism. So we must show that in addition it satisfies

$$i_D(in_{C_E}(x, f) = in_D(x, i_D \cdot f)$$

for any $f{:}\mathcal{N} \longmapsto C_E$.
Let $f = \bigvee f^n$ where f^n is defined such that for all $x \notin \mathcal{N}^n$ $f^n(x) = \Omega$ and for all $x \in \mathcal{N}^n$ $f^n(x)$ is a compact element. This is possible because C_E is an algebraic cpo. Then $in_{C_E}(x, f) = \bigvee_n in_{C_E}(x, f^n)$. It is not difficult to show that for each n

$$in_{C_E}(x, f^n) = \{[x(w).\ if\ w \in \mathcal{N}^n\ then\ f^n(w)\ else\ \Omega\ ]\}\downarrow$$

and therefore

$$in_{C_E}(x, f) = \bigvee_n \{[x(w).\ if\ w \in \mathcal{N}^n\ then\ f^n(w)\ else\ \Omega\ ]\}\downarrow .$$

So

$$\begin{aligned} i_D(in_{C_E}(x, f)) &= \bigvee_n D[\![x(w).\ if\ w \in \mathcal{N}^n\ then\ f^n(w)\ else\ \Omega\ ]\!] \\ &= \bigvee_n in_D(x, \lambda w.\ if\ w \in \mathcal{N}^n\ then\ D[\![f^n(w)]\!]\ else\ \bot\ ) \\ &= \bigvee_n in_D(x, i_D \cdot f^n) \\ &= in_D(x, i_D \cdot f). \end{aligned}$$

□

These results show that at least there are reasonable models of the language and as a byproduct we have a sound and complete proof system for the behavioural preorder. This is obtained by adding ω-induction to the proof system. Note that one can also replace the infinitary Input rule with the finitary one suggested by Proposition 3.16 and retain completeness. However CI_E, the initial fully-abstract model constructed in this section, is a term model and it would be more satisfactory if we had an independent description of it, for example as some modification of the acceptance trees in [Hen88]. The main difficulty here is to find a version of these trees which will support a reasonable definition of the restriction operator (y).

Another deficiency in this section is the general definition of what constitutes an interpretation of the language. It would be more satisfactory if this took into consideration the fact that the operator (y) also binds names. So in addition to having a special way of interpreting the input operator, using the functions in_D, we would also have a special function for restriction. One suggestion would be to have a function res_D of type $(\mathcal{N} \longmapsto D) \longmapsto D$ and then to define $D[\![(y)t]\!]\rho$ to be $res_D(\lambda y.[\![t]\!]\rho)$. With this definition α-conversion would be sound in all interpretations. However it is difficult to extend the results of this section to this new form of interpretation. It seems that a more subtle interpretation of restriction is required and one possibility is to adopt the approach taken in [Win88].

References

[BD92] M. Boreale and R. DeNicola. Testing for mobile processes. In *Proceedings of CONCUR 92*, 1992.

[GTWW77] J. Goguen, J. Thatcher, E. Wagner, and J. Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24(1):68–95, 1977.

[Gue81] I. Guessarian. *Algebraic Semantics*. Lecture Notes in Computer Science vol 99, 1981.

[Hen88] M. Hennessy. *An Algebraic Theory of Processes*. MIT Press, 1988.

[HI91] M. Hennessy and A. Ingolfsdottir. A theory of communicating processes with value-passing. *Information and Computation*, to appear, 1991.

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

[MPW92a] R. Milner, J. Parrow, and D. Walker. A calculus of mobile proceses, part i. *Information and Computation*, 100(1):1–40, 1992.

[MPW92b] R. Milner, J. Parrow, and D. Walker. A calculus of mobile proceses, part ii. *Information and Computation*, 100(1):41–77, 1992.

[Sto88] A. Stoughton. *Fully Abstract Models of Programming Languages*. Research Notes in Theoretical Computer Science, Pitman/Wiley, 1988.

[Win88] G. Winskel. A category of labelled petri nets and compositional proof system. In *Proceeding of the conference "Logic in Computer Science", Edinburgh*, July, 1988.