

# A Fully Abstract Denotational Semantics for the $\pi$ -Calculus\*

Matthew Hennessey  
School of Cognitive and Computing Sciences  
University of Sussex

June 26, 1996

## Abstract

This paper describes the construction of two set-theoretic denotational models for the  $\pi$ -calculus. The models are obtained as initial solutions to domain equations in a functor category. By associating with each syntactic construct of the  $\pi$ -calculus a natural transformation over these models we obtain two interpretations for the language.

We also show that these models are *fully abstract* with respect to natural behavioural preorders over terms in the language. By this we mean that two terms are related behaviourally if and only if their interpretations in the model are related. The behavioural preorders are the standard versions of *may* and *must* testing adapted to the  $\pi$ -calculus. S

## 1 Introduction

The  $\pi$ -calculus [15, 16] is a process algebra for describing processes which communicate by exchanging *channel names*. These names may be *private* to a particular process but such a process may decide to share a name with certain other processes by exporting it. This ability of processes to share common private channel names is the main source of the expressive power of the  $\pi$ -calculus, at least in relation to other *value-passing* process algebras.

Numerous semantic theories have been proposed for the  $\pi$ -calculus, and its variants, [16, 21, 24, 4]. However, at least until very recently, all of these theories were *behaviour* based, i.e. an operational semantics is first given to the language and then a semantic theory is developed by abstracting, using a variety of methods, from certain aspects of this operational view of processes. Thus, for example, in [21] *late* and *early* bisimulation equivalences are developed for the  $\pi$ -calculus and are characterised using a proof system for establishing identities between processes. A similar programme is carried out in [4] for an appropriate adaptation of *testing equivalence* [7].

---

\*This work was supported by the EU EXPRESS Working Group and the Royal Society. Much of the research reported here was carried out during a visit to INRIA, Sophia-Antipolis.

The use of *private* names has made it very difficult to develop set theoretic denotational models for the  $\pi$ -calculus. Their behavioural properties have much in common, at least intuitively, with that of local variables in Algol-like languages and references in ML-like languages. In papers, such as [20, 8, 22], categorical methods have been developed for providing denotational models for languages with such constructs, and recently, in [25, 19], these techniques have been applied to provide set-theoretic denotational models for fragments of the  $\pi$ -calculus, which are *fully abstract* with respect to *strong bisimulation equivalence*. The objective of this present paper is to show that similar techniques can be employed to provide set theoretic models for the  $\pi$ -calculus which generalise those presented in [12] for value-passing process calculi. In particular we provide two set theoretic denotational models for the  $\pi$ -calculus which are *fully abstract* with respect to *must* and *may* testing respectively.

The  $\pi$ -calculus consists of a small number of *constructors* for defining processes. For example  $n? \lambda x. t$  describes a process which inputs some name  $m$ , from a pre-defined set of names  $\mathcal{N}$ , along the communication channel named  $n$  and behaves according to the result of applying the abstraction  $\lambda x. t$  to the name  $m$ . Thus if  $\mathbf{P}$  is to act as a model for the  $\pi$ -calculus then to interpret this input construct we require a function  $\mathbf{in}_{\mathbf{P}}$  of type  $(\mathbf{N} \times (\mathbf{N} \Rightarrow \mathbf{P})) \Rightarrow \mathbf{P}$ , where  $\mathbf{N}$  is some appropriate representation of the set of names  $\mathcal{N}$  and we assume abstractions are interpreted in  $(\mathbf{N} \Rightarrow \mathbf{P})$ . Similarly the term  $n!m.t$  represents a process which can output the name  $m$  on the channel  $n$  and then behaves as  $t$ . To interpret this construct we require a function  $\mathbf{out}_{\mathbf{P}}$  of type  $(\mathbf{N} \times \mathbf{N} \times \mathbf{P}) \Rightarrow \mathbf{P}$ .

In addition to these input/output constructors there are also binary combinators such as  $t \parallel u$ , to run the processes  $t$  and  $u$  in parallel, and  $t + u$  to choose between  $t$  and  $u$ ; these require functions  $\parallel_{\mathbf{P}}, +_{\mathbf{P}}$  of type  $(\mathbf{P} \times \mathbf{P}) \Rightarrow \mathbf{P}$ . However the most intriguing constructor is *restriction*,  $\nu(n).t$ . For example the process  $\nu(n).m!n.t$  can output along the channel  $m$ , to a parallel process, some new private name  $k$  and this new name  $k$  may be used for subsequent communication between the parallel process and  $t[k/n]$ . Since  $\nu(n).$  is a *binding* operator it needs to be interpreted as an operator of type  $(\mathbf{N} \Rightarrow \mathbf{P}) \Rightarrow \mathbf{P}$ .

We now discuss briefly the nature of the model  $\mathbf{P}$  and how it can support the interpretation of these constructors. Following the ideas of [25], there are two important observations which underlie the nature of the model.

1. The behaviour of a process depends only on the *finite* set of channel names which it contains, or to be more precise the set of its *free* channel names, i.e. those which are not bound by the binding constructors of input and restriction. So for each finite set of names  $s$  we will have a model  $\mathbf{P}_s$  to interpret the behaviour of process terms which only use the names in  $s$ .
2. Intuitively the behaviour of a process is preserved by injective renamings of its (free) names. For example, let  $p$  be the simple process  $n!m.0 + m!n.0$  and  $\sigma$  the injective renaming which maps  $m$  to  $k$  and  $n$  to  $l$ . Then, intuitively, the behaviour of  $p\sigma$ , i.e.  $l!k.0 + k!l.0$  is determined by that of  $p$ ; it can be reconstructed from that  $p$  by means of the injection  $\sigma$ . More generally this implies the existence, for each injection  $s \mapsto^{\sigma} s'$ , of a *translation*  $\mathbf{P}_{\sigma}$  from the space of meanings  $\mathbf{P}_s$  to that of  $\mathbf{P}_{s'}$ .

The most convenient framework in which to formalise this uniform family of interpretations  $\{\mathbf{P}_s \mid s \subseteq_{fin} \mathcal{N}\}$  is in terms of functor categories. Let  $I$  be the category

whose objects are finite subsets of  $\mathcal{N}$  and whose morphisms are injections  $s \xrightarrow{\sigma} s'$ . Let  $\mathcal{D}$  be some suitable category of *domains*. Then  $\mathcal{D}^I$  is the category whose

1. objects are functors from  $I$  to  $\mathcal{D}$
2. morphisms are natural transformations

The domain  $\mathbf{P}$  we use is an object in  $\mathcal{D}^I$ , for a suitable choice of  $\mathcal{D}$ .

A particular domain  $\mathbf{P}_s$ , from this uniform family of domains, needs to be sufficiently rich to describe the behaviour of all processes with free names included in  $s$ . However it should be clear that this description will require the use of names which are not in  $s$ . For example even if the free names of the process  $p$ , of the form  $n? \lambda x.t$ , are included in  $s$  a complete description of its behaviour will have to include what happens when it receives along the channel  $n$  a name which is not in  $s$ . Denotationally this will be reflected in the fact that the domain equation characterising  $\mathbf{P}$  will have an input component of the form  $\mathbf{N} \times (\mathbf{N} \Rightarrow \mathbf{P})$ . Here  $\mathbf{N}$ , the representation of the set of names  $\mathcal{N}$  is the trivial functor whose action at  $s$  returns  $s$  itself, and  $\Rightarrow$  is the exponential operator in the functor category. This is such that the action of  $(\mathbf{N} \Rightarrow \mathbf{P})$  at  $s$  describes not only a function from  $s$  to  $\mathbf{P}_s$  but also a *uniform* collection of functions from  $s'$  to  $\mathbf{P}_{s'}$  for each  $s'$  such that  $s \xrightarrow{\sigma} s'$ . This is more than sufficient to capture the consequences of inputting names not in  $s$ .

In a similar manner the proper treatment of restriction requires that the description of  $\mathbf{P}_s$  includes names not in  $s$ . For example to describe the behaviour of the process  $\nu(x).n!x.t$ , whose free variables are included in  $s$ , one needs to describe what happens when *some* name  $y$  not in  $s$  is output on  $n$  and this name is subsequently used to interact with the process  $t[y/x]$ . Behaviourally it does not matter which  $y$  not in  $s$  we choose, and denotationally this is reflected in the fact that for any  $y, y' \notin s$ ,  $\mathbf{P}_{s \cup \{y\}}$  is isomorphic to  $\mathbf{P}_{s \cup \{y'\}}$ : the isomorphism is given by the pair of morphisms

$$\mathbf{P}_{id+y \mapsto y'}: \mathbf{P}_{s \cup \{y\}} \rightarrow \mathbf{P}_{s \cup \{y'\}} \quad \text{and} \quad \mathbf{P}_{id+y' \mapsto y}: \mathbf{P}_{s \cup \{y'\}} \rightarrow \mathbf{P}_{s \cup \{y\}}.$$

Formally,  $I$  is a symmetric monoidal category with disjoint union being the tensor,  $+$ . We use  $1$  to denote any singleton set, since all such sets are isomorphic. To reflect this behaviour the domain equation for  $\mathbf{P}$  will have an output component which includes  $\mathbf{N} \times \mathbf{s}(\mathbf{P})$  where  $\mathbf{s}$  is the functor defined so that the action of  $\mathbf{s}(A)$  at  $s$ , for any functor  $A$ , gives  $A_{s+1}$ . Intuitively this component, at  $s$ , models the output along a channel of an *arbitrary* new name.

Before giving the actual domain equation for  $\mathbf{P}$  we need to first discuss the category of domains  $\mathcal{D}$  used. Since our version of the  $\pi$ -calculus includes recursively defined processes we use *algebraic cpos*, [9]. In addition the language has operators for internal and external choice,  $\oplus$  and  $+$  respectively. With respect to the behavioural testing preorders, [10], these operators enjoy many algebraic properties. These are given in Section 2 and a domain which satisfies these laws is called a *choice domain*. For example with respect to the *must* preorder  $\oplus$  satisfies the laws of a lower semi-lattice. Consequently to model this behavioural preorder we work within the category  $l\mathcal{SL}^I$  where  $l\mathcal{SL}$  is the category whose objects are domains which are also lower semi-lattices and whose morphisms are linear continuous functions, i.e. continuous functions which also preserve the semi-lattice meet operator.  $l\mathcal{SL}^I$  has a natural exponential operator,  $\Rightarrow$ , which has a left-adjoint, a

form of tensor product which we denote by  $\otimes$ . In short  $l\mathcal{SL}^I$  is an *autonomous* category, i.e. a closed symmetric monoidal category.

We use as our domain the initial solution to the equation

$$\begin{aligned} \mathbf{P} &\cong (\mathbf{A} \times (\mathbf{N} \otimes^\top \mathbf{F}) \times (\mathbf{N} \otimes^\top \mathbf{C}))_\perp \\ \mathbf{F} &\cong \mathbf{N} \Rightarrow \mathbf{P} \\ \mathbf{C} &\cong (\mathbf{N} \otimes \mathbf{P}) + \mathbf{s}(\mathbf{P}) \end{aligned}$$

in the category  $l\mathcal{SL}^I$ . Intuitively a non-trivial object in  $\mathbf{P}$  has three components,

- $\mathbf{A}$ , a functor representing the initial nondeterministic behaviour of a process
- $\mathbf{N} \otimes^\top \mathbf{F}$ , a functor representing the input behaviour, as explained briefly above, although we use a non-standard product operator  $\otimes^\top$
- $\mathbf{N} \otimes^\top \mathbf{C}$  a functor representing the output behaviour.  $\mathbf{C}$ , representing the possibilities of output on a particular channel, has two components,  $\mathbf{N} \otimes \mathbf{P}$  representing the standard output of a name, and  $\mathbf{s}(\mathbf{P})$  representing the output of an *arbitrary* new name, as explained briefly above.

Since  $\mathbf{P}$  is an object in  $l\mathcal{SL}^I$  it automatically has an appropriate interpretation for the internal choice operator  $\oplus$ , namely the meet operation. Moreover properties of the natural transformations used to define  $\mathbf{P}$  ensure that it is a *choice domain*, thereby providing also an interpretation for the external choice operator  $+$ . We show how to interpret all of the other constructors from our version of the  $\pi$ -calculus in  $\mathbf{P}$  and prove that it is *fully abstract* with respect to the *must* testing preorder.

Essentially the same equation, only the functor  $(\ )_\perp$  is omitted, can be solved in the category  $u\mathcal{SL}^I$ , where  $u\mathcal{SL}$  is the category of *upper* semi-lattices; the internal choice,  $\oplus$ , satisfies the laws of an upper semi-lattice with respect to the *may* testing preorder. We also prove that the resulting interpretation is *fully abstract* with respect to this behavioural preorder.

We now give an outline of the remainder of the paper. In Section 2 we review the various mathematical constructions needed to describe the model. The language and its operational semantics is given in Section 3. It contains the standard  $\pi$ -calculus constructors for input, output, restriction, parallel and the empty process. In addition we allow recursive definitions of processes together with name matching and mismatching; the latter takes the form of a constructor *if  $b$  then  $t$  else  $u$*  where  $b$  can be any boolean expression from a very simple language for comparing names. As discussed above we also use an internal choice operator  $\oplus$ , which is definable in the more standard versions of the  $\pi$ -calculus. Finally in place of the *CCS* choice operator, which normally forms part of the  $\pi$ -calculus, we use external choice. Modulo the change from *CCS* choice to external choice the operational semantics, in the form of a reduction relation, is standard, [15]. This section ends with the definition of two behavioural preorders  $\sqsubseteq_{must}$  and  $\sqsubseteq_{may}$  over process terms: intuitively  $p \sqsubseteq_{must} q$  ensures that every test, in the form of another process, guaranteed by  $p$  is also guaranteed by  $q$ , while  $p \sqsubseteq_{may} q$  means that  $q$  is capable of satisfying all those tests which  $p$  can satisfy. The definitions are taken directly from the general framework presented in [10].

In the next section, Section 4 we first present the general requirements necessary for a domain  $\mathbf{P}$  to provide an interpretation for the language. This is followed by the description of the interpretation in any such domain. The main point of this section is to give the definition of a particular model, satisfying these requirements, as the initial solution to the domain equation given above in the  $l\mathcal{SL}^I$ .

Section 5 contains the main result of the paper. The model  $\mathbf{P}$  provides for all  $s$ -environments  $\rho$ , i.e. mappings from  $\mathcal{N}$  to  $s$ , an interpretation  $\llbracket p \rrbracket_{s\rho}$  of the process term  $p$  in the domain  $\mathbf{P}_s$ . We prove

$$\text{for all process terms } p, q, p\rho \sqsubseteq_{must} q\rho \text{ if and only if } \llbracket p \rrbracket_{s\rho} \leq \llbracket q \rrbracket_{s\rho},$$

i.e. the model is *fully abstract* with respect to the behavioural preorder  $\sqsubseteq_{must}$ . This section first gives an overview of the proof of this result, reducing it to three independent theorems. These are tackled in three subsequent subsections. The main technical device is that of an *acceptance*. This consists of a finite sequence of *actions*, appropriate to the  $\pi$ -calculus, followed either by  $\Omega$ , indicating an ability to diverge, or by a finite set of *communication potentials*, a communication potential taking the form either  $n?$  or  $n!$ .

- The first result is an *internal full abstraction* result for the model. Sets of acceptances can be associated with objects in the domains  $\mathbf{P}_s$  and we prove that equality in the model is determined by these sets.
- Sets of acceptances can also be associated behaviourally with process terms of the language, using the operational semantics. The second result, in Subsection 5.2, states that this behavioural set of acceptances coincides with that associated with the interpretation of the term in the model.
- The final result shows that the behavioural preorder  $\sqsubseteq_{must}$  is also determined by sets of acceptances, those associated behaviourally with process terms.

The paper ends with a brief section, Section 6, outlining similar results for *may testing*. A minor variation on the domain equation is solved in the category  $I^{u\mathcal{SL}}$ , where  $u\mathcal{SL}$  is the category of upper semi-lattices, and an outline of a proof that the resulting interpretation is fully abstract with respect to  $\sqsubseteq_{may}$ .

We end this introduction with a brief comparison with other research. As we have already stated, most of the semantic investigations into the  $\pi$ -calculus has been based on behavioural theories of processes. Those presented in [15, 16] are based on suitable adaptations of *bisimulation equivalence*, [14], and these are characterised equationally, for subsets of the  $\pi$ -calculus, in [21]. Similarly in [4, 11], the testing preorders from [7] are adapted to the  $\pi$ -calculus and equationally characterised. In [4] the standard version of the language is used while [11] uses essentially the same version as the present paper. Indeed much of the equational treatment of processes in Section 5.2, in particular their reduction to head normal forms, is taken directly from this Technical Report. However the model constructed is a *term model*, defined by quotienting process terms by the provability relation generated by the proof system and using an ideal completion construction to obtain a domain.

The models constructed in the present paper are set theoretic, in the sense that they are defined using more or less standard domain theoretic constructions, although the categories in which the domain equations are solved are rather complicated. The techniques used are borrowed directly from [25] where a fully abstract model is constructed for the recursion-free sublanguage of the  $\pi$ -calculus. The domain used is the initial solution in the category  $SFP^I$  of a domain equation which is a simple variation on the domain equation used in [1] to model *CCS* with respect to late strong bisimulation. No explicit definition is given for the natural transformations which interpret language constructs but the use of the category  $SFP^I$ , and the natural transformation  $\mathfrak{s}$  is well-motivated from a monadic view of computation, as expounded in [18].

A very similar approach is taken in [19] where a very similar model to that in [25] is constructed for the same subset of the  $\pi$ -calculus with respect to the same equivalence, strong late bisimulation equivalence, although the same model is used to characterise the associated congruence. However here the emphasis is more on exhibiting a general framework which can be instantiated to yield semantic models for a range of languages and the use of monads as a structuring mechanism for yielding denotational semantics. Much use is made of a general meta-language within which the  $\pi$ -calculus can be interpreted. In particular the proofs relating the operational and denotational semantics are expressed in terms of this meta-language.

From papers such as [8, 20, 22], it is clear that functor categories provide very useful structures within which to interpret complex operational features involving locality. The present paper, together with [25, 19], serves to emphasise that similar techniques can also play a very significant role in understanding the semantics of the  $\pi$ -calculus, or more generally process languages with local names.

## 2 Mathematical Constructions

For *domains* we use algebraic cpos in which every bounded subset has a least upper bound, [9]. A *predomain* is a domain which may lack a least element, i.e. only non-empty bounded subsets have a least upper bound. However we will generally use domains with structure. Let  $l\mathcal{SL}_p$  be the category whose objects are predomains  $D$  endowed with a continuous binary operator  $\wedge$  which satisfy the lower semi-lattice laws

$$\begin{aligned} x \wedge x &= x \\ x \wedge y &= y \wedge x \\ (x \wedge y) \wedge z &= x \wedge (y \wedge z) \\ x \wedge y &\leq x \end{aligned}$$

and whose morphisms are *linear* continuous functions, i.e. continuous functions which preserve  $\wedge$ . This category has products and coproducts; the product is ordinary cartesian product  $D \times E$ , with  $\wedge$  defined pointwise, while the coproduct  $D + E$  can be defined by  $D^\top \times E^\top - \{\top \times \top\}$ , where  $D^\top$  is the lower semi-lattice obtained from  $D$  by adjoining a new top element  $\top$ . For any set  $s$  the set of maps from  $s$  to  $D$ , where  $D$  is an object in  $l\mathcal{SL}_p$ , is also in  $l\mathcal{SL}_p$  as  $\wedge$  and the order can be defined pointwise; this object is denoted by  $(s \rightarrow D)$ . In the same way the set of morphisms between two objects,  $D_1 \Rightarrow D_2$ , is also in the category. However  $- \Rightarrow D_2$  is not right-adjoint to  $D_1 \times -$ .

Nevertheless it does have a right-adjoint, which we denote by  $D_1 \otimes -$ . This means that  $l\mathcal{SL}_p$  is naturally an autonomous category, i.e. is a closed symmetric monoidal category. The tensor product,  $D_1 \otimes D_2$ , can be constructed by quotienting formal finite meets of objects from the product  $D_1 \times D_2$  with respect to equations dictating that  $\wedge$  is linear in both arguments. Every continuous function  $f$  from  $D_1 \times D_2$  to  $D$  which is linear in both arguments determines a unique morphism  $f^\otimes : D_1 \otimes D_2 \rightarrow D$  which satisfies  $f(a, b) = f^\otimes(a \otimes b)$ . However for notational convenience we continue to use  $\pi_i$ ,  $i = 1, 2$ , as the morphisms from  $D_1 \otimes D_2$  to  $D_i$  determined by the projections from  $D_1 \times D_2$  to  $D_i$ . We will also use  $D^{\otimes k}$  to denote the product of  $k$  copies of  $D$ ,  $D \otimes \dots \otimes D$ .

Let  $\hat{s}$  denote the free lower-semi-lattice generated by a finite set  $s$ ; this can be represented as  $\mathcal{P}^{ne}(s)$ , the set of non-empty subsets of  $s$  ordered by reverse subset inclusion. Since for finite domains continuity requirements degenerate to monotonicity requirements this means that  $\hat{s}$  is also the free object in  $l\mathcal{SL}_p$  generated by the finite set  $s$ . The product  $\hat{s} \otimes \dots \otimes \hat{s}$ , can be represented by  $\mathcal{P}^{ne}(s_1 \times \dots \times s_k)$ , again ordered by reverse subset inclusion and set union as  $\wedge$ . Also for any object  $D$  in  $l\mathcal{SL}_p$ ,  $\hat{s} \otimes D$  has a simple representation as  $(s \xrightarrow{ne} D)$ , the set of non-empty partial functions from  $s$  to  $D$ ; here the order is defined by  $f \leq g$  if

- $domain(g) \subseteq domain(f)$
- for every  $n \in domain(g)$ ,  $f(n) \leq g(n)$ .

and  $f \wedge g$  is given by

$$\begin{aligned} (f \wedge g)x &= f(x) \wedge g(x), \quad x \in domain(f) \cap domain(g) \\ &= f(x), \quad x \in domain(f) - domain(g) \\ &= g(x), \quad x \in domain(g) - domain(f). \end{aligned}$$

Note that this implies  $domain(f \wedge g) = domain(f) \cup domain(g)$ . We will frequently use the construction  $(\hat{s} \otimes D)^\top$  and therefore introduce the more convenient notation  $(\hat{s} \otimes^\top D)$ ; note that this can also be represented by  $(s \xrightarrow{} D)$  the set of partial, but possibly empty, functions from  $s$  to  $D$ , with the ordering and  $\wedge$  as defined above.

As mentioned in the introduction we will interpret the  $\pi$ -calculus in domains which have even more structure. Of course to solve least fixpoints we require a least element in our domains and therefore will sometimes use the sub-category  $l\mathcal{SL}$  where the objects are domains; as usual we use  $(\ )_\perp$  to denote the functor from  $l\mathcal{SL}_p$  which adjoins a bottom element to a predomains, with *down* representing its right adjoint. Note that in objects from  $l\mathcal{SL}$ ,  $\wedge$  is always strict.

The meet operator  $\wedge$  will be used to interpret *internal choice* but we will also require an interpretation for *external choice*,  $+$ , and the empty process  $0$ . Unfortunately we can not work with distributive lattices as the operator  $+$  will not behave, operationally, as a join operator;  $p_1 + p_2$  is in general unrelated behaviorally to  $p_i$ . Nevertheless we can salvage many of the properties of distributive lattices. A *lower choice predomain* is an object in  $l\mathcal{SL}_p$  endowed with an extra continuous binary operator  $+$  and a constant  $0$  which satisfy the laws:

$$\begin{aligned} x + x &= x \\ x + y &= y + x \end{aligned}$$

$$\begin{aligned}
(x + y) + z &= x + (y + z) \\
x + \mathbf{0} &= x \\
x + (y \wedge z) &= (x + y) \wedge (x + z) \\
x \wedge (y + z) &= (x \wedge y) + (x \wedge z) \\
x \wedge y &\leq x + y
\end{aligned}$$

i.e. the extra structure is that of a commutative semi-group and both binary operators distribute over each other. Note that  $+$ , in addition to  $\wedge$ , is strict. Also the last law given above is derivable from the others but nevertheless it is convenient to include it. The initial object in the category of lower choice predomains, (where the morphisms preserve all the operators), generated by a finite discrete set  $s$  is the set of *Acceptance sets* over  $s$ ,  $\mathcal{A}(s)$ . An acceptance set over  $s$  is a finite non-empty subset,  $\mathcal{A}$ , of the powerset of  $s$  which satisfies

- $A, B \in \mathcal{A}$  implies  $A \cup B \in \mathcal{A}$
- $A, B \in \mathcal{A}, A \subseteq C \subseteq B$  implies  $C \in \mathcal{A}$ .

Note that this definition is a slight generalisation of that used in [10] as we do not require  $\cup \mathcal{A}$  to be  $s$ . In  $\mathcal{A}(s) +$  is defined pointwise,  $\mathcal{A} + \mathcal{B} = \{A \cup B \mid A \in \mathcal{A}, B \in \mathcal{B}\}$ , while  $\mathcal{A} \wedge \mathcal{B}$  is the smallest acceptance set containing both  $\mathcal{A}$  and  $\mathcal{B}$ . However we will tend to work in the more general category  $l\mathcal{SL}_p$  because some constructions will not preserve *external choice*, although all will preserve *internal choice*. Instead we will see that some constructions within  $l\mathcal{SL}_p$  will automatically lead to *choice predomains*. For example adjoining a top element to any object of  $l\mathcal{SL}_p$ , i.e. a unit for  $\wedge$ , constructs a choice predomain, since  $+$  can be defined to be  $\wedge$ .

These categories have variations, the categories  $u\mathcal{SL}_p$  and  $u\mathcal{SL}$ , obtained by considering *upper* semi-lattices in place of lower semi-lattices; that is the objects are predomains, domains respectively, endowed with a continuous binary operator  $\vee$  which satisfies the laws of an upper-semilattice:

$$\begin{aligned}
x \vee x &= x \\
x \vee y &= y \vee x \\
(x \vee y) \vee z &= x \vee (y \vee z) \\
x &\leq x \vee y
\end{aligned}$$

In objects from these categories all joins exist, since by definition all finite joins and all directed joins exist. An *upper* choice predomain is defined analogously; it is an object in  $u\mathcal{SL}_p$  with two extra continuous operators  $+$  and  $\mathbf{0}$  satisfying the laws given above. However in this case much of the algebraic structure degenerates; it is straightforward to show that in any upper choice domain  $\vee$  and  $+$  coincide. Moreover every upper choice predomain is automatically an upper choice domain as  $\mathbf{0}$  is a minimal element.

We now turn our attention to the functor categories we require. Let  $I$  denote the category whose objects are finite subsets of the set of names  $\mathcal{N}$  and whose morphisms are injections. This category has a symmetric monoidal structure; the tensor, which we denote by  $+$  is given by disjoint union and the zero by the empty set. Moreover it can be freely generated from one object  $1$ , [19], which may be taken to be any any



singleton set. Let  $l\mathcal{D}_p$  denote the category whose objects are functors from  $I$  to  $l\mathcal{S}\mathcal{L}_p$  and whose morphisms are natural transformations. There are of course the usual variations on this, for example  $l\mathcal{D}, u\mathcal{D}_p, u\mathcal{D}$ , obtained by using  $l\mathcal{S}\mathcal{L}, u\mathcal{S}\mathcal{L}_p, u\mathcal{S}\mathcal{L}$  respectively, in place of  $l\mathcal{S}\mathcal{L}_p$ . These are autonomous categories as they inherit much of the structure of the underlying categories. Products, coproducts and  $\otimes$  are defined pointwise but as usual exponentiation,  $\Rightarrow$ , the right-adjoint of  $\otimes$ , must be expressed in terms of natural transformations. The objects in  $(D \Rightarrow E)_s$ , i.e.  $(D \Rightarrow E)$  acting on the set  $s$ , are dependent families of objects

$$\prod_{s \xrightarrow{i} s'} (D_{s'} \Rightarrow E_{s'})$$

satisfying certain uniformity constraints, while the morphisms between exponentials are determined by the requirement for  $\Rightarrow$  to be functorial. Luckily we will only have a restricted need for exponentiations.

Let  $\mathcal{A}$  be the object in  $l\mathcal{D}_p$  whose action on the finite set  $s$  gives the free (lower) choice predomain  $\mathcal{A}(s + s)$ , where  $s + s$  is the disjoint union of  $s$  with itself, and which acts on the morphism  $s \xrightarrow{i} s'$  to give the obvious continuous function from  $\mathcal{A}_s$  to  $\mathcal{A}_{s'}$ , defined by  $\mathcal{A}_{(s \xrightarrow{i} s')} \mathcal{A} = \{i(a) \mid a \in \mathcal{A}\}$ . The object  $\mathbf{N}$  is defined analogously: on the set  $s$  it gives the free object of  $l\mathcal{S}\mathcal{L}_p$  generated by  $s$ , i.e. the set of subsets of  $s$ , and on morphisms it simply generalises injections to sets. To construct our model we will only require exponentials of the form  $(\mathbf{N} \Rightarrow A)$  which have a particularly simple form;  $(\mathbf{N} \Rightarrow A)_s$  can be represented by a collection of functions over  $\mathcal{N}$ .

Let  $\mathcal{F}_s(A)$  denote the set of all functions  $f$  with domain  $\mathcal{N}$  such that

1.  $f(n) \in A_{s \cup \{n\}}$
2. for all  $n, m$  not in  $s$ ,  $f(n) = A_{(id+m \mapsto n)} f(m)$ .

Under the standard pointwise ordering this is a predomain which is a representation of the predomain  $(\mathbf{N} \Rightarrow A)_s$ .

We wish to solve a domain equation in  $l\mathcal{D}$  which involves finding the initial solution to a functor over it. Note that  $l\mathcal{S}\mathcal{L}$  is a *cpo-enriched* category, i.e. in the terminology of [9] it is an  $\mathcal{O}$ -category, and this is inherited pointwise by  $l\mathcal{D}$ . This means that any *locally continuous* functor, [9], will have an initial fixpoint. In our domain equation we will use the locally continuous functors  $\times$ ,  $\otimes$  and  $\Rightarrow$  together with the very simple one  $\mathbf{s}$ . It acts on a functor  $A$  as follows:

$$\begin{aligned} (\mathbf{s} A)_s &= A_{(s+1)} \\ (\mathbf{s} A)_{(s \xrightarrow{i} s')} &= A_{(i+1)} \end{aligned}$$

where  $i + 1$  is the obvious morphism in  $I$  from  $s + 1$  to  $s' + 1$ . For any functor  $A$  we use  $up_A$  to denote the embedding of  $A$  into  $\mathbf{s}(A)$ , i.e.  $up_A$  is the natural transformation between  $A$  and  $\mathbf{s}(A)$  which at  $s$  is given by the morphism  $A_{in}$  from  $A_s$  to  $A_{(s+1)}$  where  $s \xrightarrow{in} s + 1$  denotes the injection of  $s$  into  $s + 1$ .

The functor  $\mathbf{s}$  satisfies the following useful properties:

$$\begin{aligned} \mathbf{s}(A^\top) &\simeq (\mathbf{s}(A))^\top \\ \mathbf{s}(A_\perp) &\simeq (\mathbf{s}(A))_\perp \end{aligned}$$

$$\begin{aligned}
\mathbf{s}(A \times B) &\simeq \mathbf{s}(A) \times \mathbf{s}(B) \\
\mathbf{s}(A + B) &\simeq \mathbf{s}(A) + \mathbf{s}(B) \\
\mathbf{s}(\mathbf{N} \otimes A) &\simeq (\mathbf{N} \otimes \mathbf{s}(A)) + \mathbf{s}(A) \\
\mathbf{s}(\mathbf{N} \Rightarrow A) &\simeq \mathbf{s}(\mathbf{N}) \Rightarrow \mathbf{s}(A)
\end{aligned}$$

*Notation:* In general we use  $a: A \rightarrow B$  to denote the fact that, in the category under consideration,  $a$  is a morphism from the object  $A$  to the object  $B$ . In the particular case of the underlying category  $I$  we use  $A \xrightarrow{a} B$  and sometimes when working in a functor category we will use  $a: A \dashrightarrow B$  to emphasise that the morphisms are natural transformations. We will always use  $(A \Rightarrow B)$  to indicate the exponential in a category, i.e. the object consisting of all morphisms from  $A$  to  $B$ .

### 3 The Language

Let  $\mathcal{V}$  be a set of process variables, ranged over by  $X, Y, \dots$ , and  $\mathcal{N}$  a countable set of names ranged over by  $x, y, n, m, \dots$ . Then the syntax of terms is given by:

$$\begin{aligned}
t \in Exp &::= 0 \mid n?f \mid n!x.t \mid \nu(n).t \\
&\mid t \parallel t \mid t + t \mid t \oplus t \\
&\mid \text{if } b \text{ then } t \text{ else } t \mid X \mid \text{rec } X.t \\
f \in Abs &::= \lambda x.t \\
b \in Bool &::= x = y \mid \neg b \mid b \wedge b \mid b \vee b
\end{aligned}$$

There are two binding operators for names and one for process variables. Terms may have free occurrences of both and these are determined by the simple inference system given in Figure 1. The judgements are of the form  $\Gamma \vdash t : A$  where  $\Gamma$  is a list of names and process variables which may occur free in  $t$ . The type of terms may be  $P$ , for processes or  $F$  for abstractions. As is standard we use notation such as  $\Gamma + n$  to indicate the list obtained from  $\Gamma$  by adding  $n$ , assuming that  $n$  does not already appear in  $\Gamma$ . A name substitution is a function over  $\mathcal{N}$  and we assume the standard definition of the application of a substitution  $\sigma$  to a term  $t$  to give the new term  $t\sigma$ ; this renames bound names as required in order to avoid the capture of substituted names.

The operational semantics of the language is given as a reduction relation  $\xrightarrow{\tau}$  between closed terms, terms  $p$  such that  $\underline{x} \vdash p : P$ . We will often refer to these as *process terms*, ranged over by  $p, q, \dots$ . For such terms we let  $fv(p)$  denote the set of names occurring freely in  $t$ , i.e. the names  $n$  such that  $\underline{x} \vdash p : P$  implies  $n$  occurs in  $\underline{x}$ . As usual the operational semantics is expressed using a *structural congruence*, which is defined in the third part of Figure 2. The operational semantics is then given as the least one satisfying the axioms and rules, also given in this Figure. The definition of  $\xrightarrow{\tau}$  uses two auxiliary relations,  $\xrightarrow{n?m}$ , representing the ability to input the name  $m$  on channel  $n$  and  $\xrightarrow{n!m}$  representing the ability to output  $m$  along  $n$ . In Figure 2  $\mu$  ranges over the three possible types of action  $\tau$ ,  $n?m$ , and  $n!m$ . Note that  $\alpha$ -equivalence, defined in the standard manner, is included in the structural congruence and therefore operationally we work modulo  $\alpha$ -equivalence. In the rule for the **if then else** statement we also assume

$$\begin{array}{c}
\frac{}{\underline{x}; X_1 \dots X_i \dots X_n \vdash X_i : P} \quad \frac{\dots \alpha \beta \dots \vdash t : P}{\dots \beta \alpha \dots \vdash t : P} \\
\\
\frac{}{\Gamma \vdash 0 : P} \quad \frac{\Gamma + n \vdash t : P}{\Gamma \vdash \nu(n).t : P} \\
\\
\frac{\Gamma + n \vdash f : F}{\Gamma + n \vdash n? f : P} \quad \frac{\Gamma + n + x \vdash t : P}{\Gamma + n + x \vdash n!x.t : P} \\
\\
\frac{\Gamma \vdash t : P \quad \Gamma \vdash u : P}{\Gamma \vdash t \text{ op } u : P} \quad \text{op} = \parallel, +, \oplus \\
\\
\frac{\Gamma \vdash t : P \quad \Gamma \vdash u : P}{\Gamma \vdash \text{if } b \text{ then } t \text{ else } u : P} \\
\\
\frac{\Gamma + x \vdash t : P}{\Gamma \vdash \lambda x.t : F} \quad \frac{\Gamma + X \vdash t : P}{\Gamma \vdash \text{rec } X.t : P}
\end{array}$$

Figure 1: Free variables in terms

some evaluation mechanism for boolean expressions, which in view of their simplicity, should be obvious.

As an example of an application of the reduction rules consider the process term

$$p = (\nu(y).n!y.r + n!x.r') \parallel n?(\lambda z.z!y.0 + z!x.0)$$

Up to  $\alpha$ -equivalence there are two possible reductions from  $p$ , the first to  $r' \parallel (x!y.0 + x!x.0)$  by the transmission of the name  $x$ , and the second to  $\nu(w).r[w/y] \parallel (w!y.0 + w!x.0)$  where  $w$  is some new name.

We refer the reader to papers such as [23, 17] for evidence of the expressive power of this language. Here we confine our attention to defining two behavioural preorders over process terms and in later sections we build fully abstract denotational models for these preorders.

The behavioural preorders are based directly on the ideas of *testing* as developed for example in [7, 10]. A process  $p$  is tested (to conform to some behavioural requirement) by running it in parallel with another process  $e$ , presumably designed with the behavioural requirement in mind. This test is deemed a success if the experimenter,  $e$ , reaches a specified state. To model this *success state* we introduce a *new* name  $\omega$ , not occurring in  $\mathcal{N}$  and we say  $e$  is in the success state if it can emit an output along  $\omega$ , we indicate this by the notation  $e \xrightarrow{\omega}$

**Definition 1** *Let  $p$  must  $e$  if for every maximal computation*

$$e \parallel p = c_0 \xrightarrow{\tau} c_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} c_k \xrightarrow{\tau} \dots$$

Axioms:

$$\frac{}{n ? \lambda x. p \xrightarrow{n!m} p[m/x]} \quad \frac{}{n ! m . p \xrightarrow{n!m} p}$$

$$\frac{}{p \oplus q \xrightarrow{\tau} p} \quad \frac{}{p \oplus q \xrightarrow{\tau} q}$$

$$\frac{}{\text{rec } X . t \xrightarrow{\tau} t[\text{rec } X . t / X]}$$

Communication Rule:

$$\frac{p \xrightarrow{n!m} p', q \xrightarrow{n!m} q'}{p \parallel q \xrightarrow{\tau} p' \parallel q'} \quad \frac{p \xrightarrow{n!m} p', q \xrightarrow{n!m} q'}{p \parallel q \xrightarrow{\tau} p' \parallel q'}$$

Context Rules:

$$\frac{p \xrightarrow{\mu} p'}{p \parallel q \xrightarrow{\mu} p' \parallel q} \quad \frac{q \xrightarrow{\mu} q'}{p \parallel q \xrightarrow{\mu} p \parallel q'}$$

$$\frac{p \xrightarrow{\tau} p'}{p + q \xrightarrow{\tau} p' + q} \quad \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'} \quad \alpha = n?m, n!m$$

$$\frac{p \xrightarrow{\mu} p'}{\nu(x) . p \xrightarrow{\mu} \nu(x) . p'} \quad x \text{ not in } \mu$$

$$\frac{[[b]] = \text{true}, p \xrightarrow{\mu} p'}{\text{if } b \text{ then } p \text{ else } q \xrightarrow{\mu} p'} \quad \frac{[[b]] = \text{false}, q \xrightarrow{\mu} q'}{\text{if } b \text{ then } p \text{ else } q \xrightarrow{\mu} q'}$$

$$\frac{p \equiv p', p' \xrightarrow{\mu} q', q' \equiv q}{p \xrightarrow{\tau} q}$$

Structural Equivalence:

$$(\nu(x) . p) \parallel q \equiv \nu(x) . (p \parallel q) \text{ provided } x \notin \text{fv}(q)$$

$$p \equiv q \text{ provided } p \equiv_{\alpha} q$$

$$(p \parallel q) \parallel r \equiv p \parallel (q \parallel r)$$

$$p + q \equiv q + p$$

Figure 2: Operational Semantics

there is some  $n$  such that  $c_n \xrightarrow{\omega}$

Then  $p \sqsubseteq_{\text{must}} q$  if for every test  $e$ ,  $p$  must  $e$  implies  $q$  must  $e$ .

This preorder is based on the idea of a process guaranteeing certain behaviour. There is a weaker preorder which captures the idea of processes being capable of certain be-

haviour:  $p \sqsubseteq_{may} q$  if for every test  $e$ ,  $p \text{ may } e$  implies  $q \text{ may } e$ , where  $q \text{ may } e$  means that there is some maximal computation from  $e \parallel q$  which reaches a state  $c_n$  such that  $c_n \xrightarrow{\omega}$ .

In the next Section we build a model which provides a denotational semantics which is fully abstract with respect to  $\sqsubseteq_{must}$  and in Section 6 we show how this can be modified so as to provide a semantics which is fully abstract with respect to  $\sqsubseteq_{may}$ .

We end this section by noting that both of these preorders are natural generalisations of Morris style contextual preorders, [3, 13]. We have restricted the possible testing contexts to be of the simple form  $e \parallel [ \ ]$ . We conjecture that the model can also be used to provide a denotational semantics which is fully abstract with respect to a notion of testing where more general contexts are allowed, although it is essential that the action for reporting the success  $\omega$  be new, i.e. an action which cannot be performed by the process being tested. A result along these lines, for *may* testing, is given at the end of Section 6.

## 4 The Interpretation

We interpret the language in the category  $\mathcal{LD}_p$ . To do so we use an object in the subcategory  $\mathcal{LD}$ , which we call  $\mathbf{P}$ , together with an appropriate morphism, i.e. a natural transformation, for each combinator in the language. In the standard way, [6], we can then associate with each typing judgement  $\underline{x} \vdash p : P$  a morphism from  $\mathbf{N}^{\otimes \underline{x}}$  to  $\mathbf{P}$  in the category  $\mathcal{LD}_p$ . Because of the nature of the objects  $\mathbf{N}^{\otimes \underline{x}}$  these morphisms have a simple representation in terms of *environments*. For any finite subset  $s$  of  $\mathcal{N}$  let  $ENV_s$  be the set of mappings from  $\mathcal{N}$  to  $s$ . For  $\rho, \rho' \in ENV_s$  let  $\rho =_{\underline{x}} \rho'$  if  $\rho(y) = \rho'(y)$  for every name  $y$  in the vector  $\underline{x}$ . Then the set of morphisms from  $(\mathbf{N}^{\otimes \underline{x}})_s$  to  $\mathbf{P}_s$ , in the category  $\mathcal{LSL}_p$ , is in one-one correspondence with the set of mappings from  $ENV_s$  to  $\mathbf{P}_s$  which preserve  $=_{\underline{x}}$ . In other words our semantics is equivalent to a family of interpretations

$$\llbracket \ ]_s : ENV_s \rightarrow \mathbf{P}_s$$

Moreover naturality ensures that this is a *uniform* family of interpretations; the family of domains  $\{\mathbf{P}_s \mid s \subseteq \mathcal{N}\}$  comes equipped with translation morphisms  $(\mathbf{P}_\sigma) : \mathbf{P}_s \rightarrow \mathbf{P}_{s'}$ , for each injection  $s \xrightarrow{\sigma} s'$  and these translation morphisms also relate the interpretation of terms in the different domains:

$$\llbracket p \rrbracket_{s'}(\sigma \circ \rho) = (\mathbf{P}_\sigma)(\llbracket p \rrbracket_s \rho).$$

In order to give this interpretation we require, in addition to the object  $\mathbf{P}$ ,

- $\oplus_{\mathbf{P}}$ , a morphism from  $\mathbf{P} \otimes \mathbf{P}$  to  $\mathbf{P}$ , to interpret internal choice.  $\mathbf{P}$  will be a lower semi-lattice and  $\oplus_{\mathbf{P}}$  be the meet operation  $\wedge^{\otimes}$ .
- $+_{\mathbf{P}}$ , a morphism from  $\mathbf{P} \otimes \mathbf{P}$  to  $\mathbf{P}$ , to interpret external choice. The lower semi-lattice  $\mathbf{P}$  will be constructed in such a way that it is also a *choice domain* and will therefore come equipped with a natural candidate for  $+_{\mathbf{P}}$ , namely the extra binary operator  $+^{\otimes}$ .
- $0_{\mathbf{P}}$ , an object in  $\mathbf{P}$  to interpret the language construct 0. Again we will have a natural candidate since  $\mathbf{P}$  will be a *choice domain*.

- $\llbracket \mathbf{P} \rrbracket$ , a morphism from  $\mathbf{P} \otimes \mathbf{P}$  to  $\mathbf{P}$  to model the parallel construct.
- $\nu_{\mathbf{P}}$ , a morphism from  $(\mathbf{N} \Rightarrow \mathbf{P})$  to  $\mathbf{P}$  to interpret the construct  $\nu(n).t$ .
- Abstractions will be interpreted in  $(\mathbf{N} \Rightarrow \mathbf{P})$  and therefore to interpret the language construct  $n? f$  we need a morphism  $\mathbf{in}_{\mathbf{P}}$  from  $\mathbf{N} \otimes (\mathbf{N} \Rightarrow \mathbf{P})$  to  $\mathbf{P}$ .
- In order to interpret the output construct  $n!x.t$  we require a morphism  $\mathbf{out}_{\mathbf{P}}$  from  $\mathbf{N} \otimes \mathbf{N} \otimes \mathbf{P}$  to  $\mathbf{P}$ .

Because of the presence of recursive terms the definition of the semantics will also apply to open terms. So in fact we interpret the general type judgement  $\Gamma \vdash t : P$  as a morphism from  $\llbracket \Gamma \rrbracket$  to  $\mathbf{P}$  where the object  $\llbracket \Gamma \rrbracket$  is determined by the structure of  $\Gamma$ . If it has the form  $\underline{x}; \underline{X}$  then it has the form  $\mathbf{N}^{\otimes \underline{x}} \otimes \mathbf{P}^{\otimes \underline{X}}$ . Therefore when  $t$  is a closed term this will be a morphism from  $\mathbf{N}^{\otimes \underline{x}}$  to  $\mathbf{P}$ , as promised above. The definition of  $\llbracket \Gamma \vdash t : P \rrbracket$  is by induction on the inference of this judgement:

- $\llbracket \underline{x}; X_1 \dots X_i \dots X_n \vdash X_i : P \rrbracket = \pi_i \circ \pi_2$
- $\llbracket \dots x_{i+1} x_i \dots; \underline{X} \vdash t : P \rrbracket = \llbracket \dots x_i x_{i+1} \dots; \underline{X} \vdash t : P \rrbracket \circ \langle \langle \pi_1 \otimes \dots \otimes \pi_{i+1} \otimes \pi_i \dots \otimes \pi_n \rangle \otimes \pi_2 \rangle$
- $\llbracket \underline{x}; \dots X_{i+1} X_i \dots \vdash t : P \rrbracket = \llbracket \underline{x}; \dots X_i X_{i+1} \dots \vdash t : P \rrbracket \circ \langle \pi_1 \otimes \langle \pi_1 \otimes \dots \otimes \pi_{i+1} \otimes \pi_i \dots \pi_n \rangle \rangle$
- $\llbracket \Gamma \vdash 0 : P \rrbracket = !_{\llbracket \Gamma \rrbracket} \circ 0_{\mathbf{P}}$
- $\llbracket \Gamma + n \vdash n? f : P \rrbracket = \mathbf{in}_{\mathbf{P}} \circ \langle (\pi_n \circ \pi_1) \otimes \llbracket \Gamma + n \vdash f : F \rrbracket \rangle$
- $\llbracket \Gamma + n + x \vdash n!x.t : P \rrbracket = \mathbf{out}_{\mathbf{P}} \circ \langle (\pi_n \circ \pi_1) \otimes (\pi_x \circ \pi_1) \otimes \llbracket \Gamma + n + x \vdash t : P \rrbracket \rangle$
- For  $op = \parallel, +, \oplus$ ,  $\llbracket \Gamma \vdash t op u : P \rrbracket = \mathbf{P}_{op} \circ \langle \llbracket \Gamma \vdash t : P \rrbracket \otimes \llbracket \Gamma \vdash u : P \rrbracket \rangle$
- $\llbracket \Gamma \vdash \text{if } b : t \text{ then } u \text{ else } P \rrbracket = \mathbf{if} \circ \langle \llbracket \Gamma \vdash b \rrbracket \times \llbracket \Gamma \vdash t : P \rrbracket \times \llbracket \Gamma \vdash u : P \rrbracket \rangle$   
Here we assume the obvious semantic interpretation of booleans,  $\llbracket \Gamma \vdash b \rrbracket$  in  $(\llbracket \Gamma \rrbracket \rightarrow \mathbf{Bool})$ , where  $\mathbf{Bool}$  is the functor whose action at every  $s$  gives the set of booleans  $\{true, false\}$ . Then  $\mathbf{if}$  also acts pointwise; at every  $s$  it selects the second argument if the first is *true* and the third otherwise.
- $\llbracket \Gamma \vdash \lambda x.t : F \rrbracket = \lambda_{\mathbf{N}}(\llbracket \Gamma + x \vdash t : P \rrbracket)$   
Note  $\llbracket \Gamma + x \vdash t : P \rrbracket$  is a morphism in  $(\llbracket \Gamma \rrbracket \otimes \mathbf{N}) \rightarrow \mathbf{P}$  and since  $\otimes$  is left adjoint to  $\Rightarrow$  we have the mapping  $\lambda_{\mathbf{N}}$  from  $(\llbracket \Gamma \rrbracket \otimes \mathbf{N}) \rightarrow \mathbf{P}$  to  $\llbracket \Gamma \rrbracket \rightarrow (\mathbf{N} \Rightarrow \mathbf{P})$ .
- $\llbracket \Gamma \vdash \nu(n).t : P \rrbracket = \nu_{\mathbf{P}} \circ \lambda_{\mathbf{N}}(\llbracket \Gamma + n \vdash t : P \rrbracket)$
- $\llbracket \Gamma \vdash \text{rec } X.t : P \rrbracket = \mathbf{Y} \circ \lambda_{\mathbf{P}}(\llbracket \Gamma + X \vdash t : P \rrbracket)$   
Here  $\llbracket \Gamma + X \vdash t : P \rrbracket$  is a morphism in  $(\llbracket \Gamma \rrbracket \otimes \mathbf{P}) \rightarrow \mathbf{P}$  and using the adjunction between  $\otimes$  and  $\Rightarrow$  once more there is a mapping  $\lambda_{\mathbf{P}}$  from  $(\llbracket \Gamma \rrbracket \otimes \mathbf{P}) \rightarrow \mathbf{P}$  to  $\llbracket \Gamma \rrbracket \rightarrow (\mathbf{P} \Rightarrow \mathbf{P})$ . The fixpoint operator used here,  $\mathbf{Y}$ , is a morphism from  $(\mathbf{P} \Rightarrow \mathbf{P})$  to  $\mathbf{P}$ .

To complete our description of the interpretation we must exhibit the specific choice domain  $\mathbf{P}$ , an object in  $\mathcal{ID}$ , together with the required morphisms.  $\mathbf{P}$  is essentially a functorial version of the *Acceptance Tree* model of [12], adapted to the  $\pi$ -calculus.

We let  $\mathbf{P}$  be the initial solution to the equation

$$\begin{aligned}\mathbf{P} &\cong (\mathbf{A} \times (\mathbf{N} \otimes^\top \mathbf{F}) \times (\mathbf{N} \otimes^\top \mathbf{C}))_\perp \\ \mathbf{F} &\cong \mathbf{N} \Rightarrow \mathbf{P} \\ \mathbf{C} &\cong (\mathbf{N} \otimes \mathbf{P}) + \mathbf{s}(\mathbf{P})\end{aligned}$$

In order to explain the intuition behind this equation let us consider the action of this functor on an arbitrary object of  $I$ , a finite subset  $s$  of  $\mathcal{N}$ .  $\mathbf{P}_s$  is either the bottom element  $\perp$  or has three components:

- $\mathbf{A}_s$ , an acceptance set over  $s+s$ , representing the initial nondeterministic behaviour of a process. We require two copies of  $s$  in order to record both the input and output potentials of processes. To emphasise this use of  $s+s$  we let  $n?$ ,  $n!$  denote  $in_l(n)$ ,  $in_r(n)$  respectively, when applied to  $s+s$ .
- $(\mathbf{N} \otimes^\top \mathbf{F})_s$ , a finite partial function from  $s$  to  $\mathbf{F}_s$ , representing the potential input behaviour of a process on some finite set of input channels. The objects in  $\mathbf{F}_s$  represent the functional behaviour of the process on receipt of an input.  $\mathbf{F}$  is isomorphic to  $(\mathbf{N} \Rightarrow \mathbf{P})$  and so, as outlined in Section 2,  $\mathbf{F}_s$  is isomorphic to  $\mathcal{F}_s(\mathbf{P})$ . This encodes the subsequent behaviour of the process on receipt of any name in  $s$  and on receipt of an *arbitrary* name not in  $s$ .
- $(\mathbf{N} \otimes^\top \mathbf{C})_s$  another partial function from  $s$  to  $\mathbf{C}_s$ , representing the potential output behaviour along a finite number of output channels. The output behaviour associated with each channel, in  $\mathbf{C}_s$ , is of two kinds:
  - $(\mathbf{N} \otimes \mathbf{P})_s$ , a finite non-empty function from  $s$  to  $\mathbf{P}_s$ , representing a finite set of pairs, each pair consisting of a name to be output along the channel and an element of  $\mathbf{P}_s$  encoding the subsequent behaviour of the process
  - $\mathbf{s}(\mathbf{P})_s$ , i.e.  $\mathbf{P}_{(s+1)}$ , representing the subsequent behaviour of the process after outputting an *arbitrary* new name not in  $s$  along the channel.

For convenience, whenever appropriate, we will elide the isomorphism pair *unfold* and *fold* between  $\mathbf{P}$  and  $(\mathbf{A} \times (\mathbf{N} \otimes^\top \mathbf{F}) \times (\mathbf{N} \otimes^\top \mathbf{C}))_\perp$ . First note that since  $\mathbf{P}$  is in the category  $\mathcal{ID}$  it is a lower semi-lattice, thus providing an interpretation for  $\oplus$ . Moreover it is also a lower choice domain since

- $\mathbf{A}$  is a lower choice domain
- both  $\mathbf{N} \otimes^\top \mathbf{F}$  and  $\mathbf{N} \otimes^\top \mathbf{C}$  are lower semi-lattices with units for  $\wedge$ , namely  $\top$ , and therefore are also lower choice domains
- lower choice domains are closed under product and lifting.

We therefore have the required interpretation for the language operators  $0$  and  $+$ .

To define the natural transformation  $\mathbf{in}_{\mathbf{P}}: \mathbf{N} \otimes (\mathbf{N} \Rightarrow \mathbf{P}) \rightarrow \mathbf{P}$  we must define for each  $s$  a morphism  $(\mathbf{in}_{\mathbf{P}})_s$  from  $(\mathbf{N} \otimes (\mathbf{N} \Rightarrow \mathbf{P}))_s$ , i.e.  $s \xrightarrow{ne} (\mathbf{N} \Rightarrow \mathbf{P})_s$ , to  $\mathbf{P}_s$ . This is defined as follows:

$$(\mathbf{in}_{\mathbf{P}})_s g = \wedge \{ (\{ \{ n? \} \} \times (n \mapsto gn) \times \emptyset)_{\perp} \mid n \in \text{domain}(g) \}.$$

Intuitively  $(\mathbf{in}_{\mathbf{P}})_s$  is determined its action on the functions with singleton domains. So for example when applied to the function  $(n \mapsto f)$  it constructs the object  $(\{n?\} \times (n \mapsto f) \times \emptyset)_{\perp}$ .

The output natural transformation  $\mathbf{out}_{\mathbf{P}}$  is defined in a similar manner.  $(\mathbf{out}_{\mathbf{P}})_s$  is the morphism from  $(\mathbf{N} \otimes \mathbf{N} \otimes \mathbf{P})_s$ , i.e.  $(s \times s \xrightarrow{ne} \mathbf{P}_s)$ , to  $\mathbf{P}_s$  defined by

$$(\mathbf{out}_{\mathbf{P}})_s g = \wedge \{ (\{ \{ n! \} \} \times \emptyset \times (n \mapsto (m \mapsto g(n, m))))_{\perp} \mid (n, m) \in \text{domain}(g) \}.$$

The definitions of the remaining two morphisms,  $\|\mathbf{P}$  and  $\nu_{\mathbf{P}}$  are somewhat complicated. The definition of the first is not of great interest as it is determined by the requirement that the interleaving law in Figure 4 be satisfied. It is therefore relegated to Appendix A. However the entire structure of  $\mathbf{P}$ , and its construction in the functor category, is motivated by the search for an adequate definition of  $\nu_{\mathbf{P}}$ . This is given in terms of an auxiliary natural transformation  $\mathbf{new}: \mathbf{s}(\mathbf{P}) \rightarrow \mathbf{P}$ . Given  $\mathbf{new}$  we can define the morphism  $\nu_{\mathbf{P}}$ , i.e. a natural transformation from  $(\mathbf{N} \Rightarrow \mathbf{P})$  to  $\mathbf{P}$ , using a *coercion* natural transformation between  $(\mathbf{N} \Rightarrow \mathbf{P})$  and  $\mathbf{s}(\mathbf{P})$ . Let  $\mathbf{c}: (\mathbf{N} \Rightarrow \mathbf{P}) \rightarrow \mathbf{s}(\mathbf{P})$  be given by the family of morphisms  $\mathbf{c}_s$  from  $(\mathbf{N} \Rightarrow \mathbf{P})_s$ , i.e.  $\mathcal{F}_s(\mathbf{P})$ , to  $(\mathbf{s}(\mathbf{P}))_s$ , i.e.  $\mathbf{P}_{(s+1)}$ , defined by  $\mathbf{c}_s f = f(y)$  for any  $y \notin s$ . Note that this definition does not depend on the particular value of  $y$  chosen. Then the required natural transformation  $\nu_{\mathbf{P}}: (\mathbf{N} \Rightarrow \mathbf{P}) \rightarrow \mathbf{P}$  is given by  $\mathbf{new} \circ \mathbf{c}$ .

To define the natural transformation  $\mathbf{new}$  from  $\mathbf{s}(\mathbf{P})$  to  $\mathbf{P}$  we make use of the properties of  $\mathbf{s}$  given in Section 2. Using these one can easily show that  $\mathbf{s}(\mathbf{P})$  is isomorphic to

$$(\mathbf{s}(\mathbf{A}) \times (\mathbf{s}(\mathbf{N} \otimes \mathbf{F}))^{\top} \times (\mathbf{s}(\mathbf{N} \otimes \mathbf{C}))^{\top})_{\perp}.$$

Therefore the definition of  $\mathbf{new}$  will take the form  $(\mathbf{new}_1 \times \mathbf{new}_2 \times \mathbf{new}_3)_{\perp}$  where

- $\mathbf{new}_1: \mathbf{s}(\mathbf{A}) \rightarrow \mathbf{A}$
- $\mathbf{new}_2: (\mathbf{s}(\mathbf{N} \otimes \mathbf{F}))^{\top} \rightarrow \mathbf{N} \otimes^{\top} \mathbf{F}$
- $\mathbf{new}_3: (\mathbf{s}(\mathbf{N} \otimes \mathbf{C}))^{\top} \rightarrow \mathbf{N} \otimes^{\top} \mathbf{F}$

In fact  $\mathbf{new}$  will be used in the definition of  $\mathbf{new}_i$  and therefore we will take  $\mathbf{new}$  to be the *least* morphism to satisfy its definition. We define each of its components in turn.

- Intuitively the action of  $\mathbf{new}_1$  at the set  $s$  is to pointwise restrict subsets of  $\mathcal{P}(s+1)$  to subsets of  $\mathcal{P}(s)$ . Formally  $(\mathbf{new}_1)_s$  is defined by  $(\mathbf{new}_1)_s(\mathcal{A}) = \{ A \setminus s \mid A \in \mathcal{A} \}$  where  $A \setminus s = \{ n?, n! \in A \mid n \in s \}$ .
- To define  $\mathbf{new}_2$  we use the isomorphism

$$\mathbf{s}(\mathbf{N} \otimes \mathbf{F}) \simeq (\mathbf{N} \otimes \mathbf{s}(\mathbf{F})) + \mathbf{s}(\mathbf{F}).$$

If  $\mathbf{new}_f$  is a natural transformation from  $\mathbf{s}(\mathbf{F})$  to  $\mathbf{F}$  then  $(id_{\mathbf{N}} \otimes^{\top} \mathbf{new}_f) + \lambda x. \top$  is a natural transformation from  $\mathbf{s}(\mathbf{N} \otimes \mathbf{F})$  to  $\mathbf{N} \otimes^{\top} \mathbf{F}$  and  $\mathbf{new}_2$  will be defined



as its trivial extension. So to complete the definition we must specify  $\mathbf{new}_f$ . Since  $\mathbf{F}$  is isomorphic to  $\mathbf{N} \Rightarrow \mathbf{P}$ , by definition, and we have the isomorphism  $\mathbf{s}(\mathbf{N} \Rightarrow \mathbf{P}) \simeq (\mathbf{s}(\mathbf{N}) \Rightarrow \mathbf{s}(\mathbf{P}))$  we can define  $\mathbf{new}_f$  to be  $(up_{\mathbf{N}} \Rightarrow \mathbf{new})$ .

- $\mathbf{new}_3$  is defined in a similar manner as the the trivial extension of  $id_{\mathbf{N}} \otimes^{\top} \mathbf{new}_c + \lambda x. \top$  where  $\mathbf{new}_c$  is a natural transformation from  $\mathbf{s}(\mathbf{C})$  to  $\mathbf{C}$ .  $\mathbf{s}(\mathbf{C})$  is isomorphic to  $(\mathbf{N} \otimes \mathbf{s}(\mathbf{P})) + \mathbf{s}(\mathbf{P}) + \mathbf{s}(\mathbf{s}(\mathbf{P}))$  and therefore  $\mathbf{new}_3$  can be defined to be  $(id_{\mathbf{N}} \otimes \mathbf{new}) + id_{\mathbf{s}(\mathbf{P})} + \mathbf{s}(\mathbf{new}) \circ \mathit{switch}_{\mathbf{P}}$ , where  $\mathit{switch}_{\mathbf{P}}$  is the natural transformation over  $\mathbf{s}(\mathbf{s}(\mathbf{P}))$  which at  $s$  is given by the morphism  $\mathbf{P}(id_s + \mathit{switch})$ , where  $\mathit{switch}: 1 + 1 \rightarrow 1 + 1$  is the morphism in  $I$  which switches its arguments.

This completes our description of the interpretation of the  $\pi$ -calculus.

We have associated with each term  $p$  such that  $\Gamma \vdash p : P$  a natural transformation between  $\llbracket \Gamma \rrbracket$  and  $\mathbf{P}$ . As explained at the beginning of this section this gives, for each process term  $p$ , a family of interpretations  $\llbracket p \rrbracket_s$  which can be viewed as mappings from  $ENV_s$  to  $\mathbf{P}_s$ . Specifically if  $x_1 \dots x_k \vdash p : P$  then for  $\sigma \in ENV_s$  we let  $\llbracket p \rrbracket_{s\sigma}$  denote the element in  $\mathbf{P}_s$  given by  $\llbracket p \rrbracket_s(\sigma(x_1) \otimes \dots \otimes \sigma(x_k))$ . With this notation the interpretation satisfies the well known *Substitution Lemma*:

**Lemma 4.1**  $\llbracket p\sigma \rrbracket_{s\rho} = \llbracket p \rrbracket_s(\rho \circ \sigma)$ . □

## 5 Full Abstraction

In this section we give the relationship between the operational and denotational semantics. Recall from the previous section that the denotational semantics gives rise to a family of interpretations,

$$\llbracket \cdot \rrbracket_s : ENV_s \rightarrow \mathbf{P}_s,$$

one for each finite subset  $s$  of  $\mathcal{N}$ . Also each  $\rho \in ENV_s$  can act as a substitution over processes. With this notation in mind the main result of this paper can be stated as:

**Theorem 5.1** *For all process terms  $p, q$  and for every  $\rho \in ENV_s$ ,  $p\rho \sqsubseteq_{must} q\rho$  if and only if  $\llbracket p \rrbracket_{s\rho} \leq \llbracket q \rrbracket_{s\rho}$ .*

We first give an outline of the structure of the proof of this result. This is followed by three subsections which contain the details. The essential ingredient of the proof is the notion of *acceptances*. Let  $\mathbf{Acc}$  be the least set such that

- $\Omega \in \mathbf{Acc}$
- $A \in \mathbf{Acc}$  if  $A$  is a finite subset of  $Ind = \{n?, n! \mid n \in \mathcal{N}\}$
- $\mathbf{a} \in \mathbf{Acc}$  implies
  - $n?x.\mathbf{a} \in \mathbf{Acc}$
  - $n!x.\mathbf{a} \in \mathbf{Acc}$
  - $\nu(x).n!x.\mathbf{a} \in \mathbf{Acc}$

Informally these can be viewed as terms in the language (with the added constant  $\Omega$ ); viewed in this manner  $\nu(\mathbf{x}) .$  acts as a binding operator on *acceptances* and we will only consider them up to  $\alpha$ -conversion.

*Acceptances* are ordered as follows:

- $\Omega \ll \mathbf{a}$  for every *acceptance*  $\mathbf{a}$
- $\mathbf{a} \ll \mathbf{b}$  implies
  - $n? \mathbf{x} . \mathbf{a} \ll n? \mathbf{x} . \mathbf{b}$
  - $n! \mathbf{x} . \mathbf{a} \ll n! \mathbf{x} . \mathbf{b}$
  - $\nu(\mathbf{x}) . \mathbf{a} \ll \nu(\mathbf{y}) . \mathbf{b}[\mathbf{y}/\mathbf{x}]$  for every  $y \notin fv(\nu(x) . a)$ .

This ordering is lifted to sets of *acceptances* by:

$A \ll B$  if for every  $\mathbf{b} \in B$  there exists some  $\mathbf{a} \in A$  such that  $\mathbf{a} \ll \mathbf{b}$ .

Sets of *acceptances* can be associated with elements of the domains  $\mathbf{P}_s$  and, behaviourally, with process terms of the language. We first consider the latter and to do so we need some notation. The operational semantics is given in terms of a reduction relation  $\xrightarrow{\tau}$  between process terms. Here we need to describe the *potential actions* a term can perform. There are three kinds of potential actions,

1. an input action  $\xrightarrow{n?x}$ , as described in Section 3
2. an output action  $\xrightarrow{n!x}$ , also as described in Section 3
3. a *bound* output action  $\xrightarrow{\nu(z) . n!z}$

We use  $Act$ , ranged over by  $\alpha$ , to denote this set of potential actions. The bound output action may be defined by:

$$p \xrightarrow{\nu(z) . n!z} q, \text{ if } p \parallel n? \lambda y . y ! y . 0 \xrightarrow{\tau} \nu(z) . (q \parallel z ! z . 0), \text{ where } z \notin fv(p).$$

These are extended to sequences over  $Act$  by

- $p \xRightarrow{\varepsilon} q$  if  $p \xrightarrow{\tau}^* q$
- $p \xRightarrow{\alpha . s} q$  if  $p \xrightarrow{\tau}^* \xrightarrow{\alpha} \xRightarrow{s} q$

We write  $p \uparrow$  if  $p$  *diverges*, i.e. there is an infinite sequence of derivations

$$p \xrightarrow{\tau} p_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} p_k \xrightarrow{\tau} \dots$$

and  $p \downarrow$  if it *converges*, i.e. there is no such sequence. Finally let  $S(p)$  denote the subset of the set of *indications*,  $Ind$ , defined by

$$\{ n? \mid p \xRightarrow{n?x} \text{ for some } x \} \cup \{ n! \mid p \xRightarrow{n!x} \text{ or } p \xRightarrow{\nu(x) . n!x} \text{ for some } x \}.$$

With this notation we can now define the behavioural acceptances associated with a process term.

- $p \models^b \Omega$  if  $p \uparrow$

- $p \models^b A$  if  $p \Downarrow$  and  $A = S(q)$  for some  $q$  such that  $p \xrightarrow{\varepsilon} q$
- $p \models^b \alpha.a$  if  $p \Downarrow$  and  $q \models^b \alpha$  for some  $q$  such that  $p \xrightarrow{\alpha} q$

Let  $\text{Acc}^b(p)$  denote the set  $\{a \mid p \models^b a\}$ . These sets characterise the behavioural preorder  $\sqsubseteq_{\text{must}}$ :

**Theorem 5.2**  $p \sqsubseteq_{\text{must}} q$  if and only if  $\text{Acc}^b(p) \ll \text{Acc}^b(q)$ .

The proof of this theorem is given in Sub-section 5.3.

In a similar manner we can associate with each element  $d$  of the domain  $\mathbf{P}_s$  a set of acceptances  $\text{Acc}_s(d)$ ; the details are postponed until Sub-section 5.1 because the structure of  $\mathbf{P}_s$  needs to be elaborated.

The following theorem is often called an *internal full abstraction* result as it gives an alternative characterisation of identity in the model.

**Theorem 5.3** For every  $d, e \in \mathbf{P}_s$ ,  $d \leq e$  if and only if  $\text{Acc}_s(d) \ll \text{Acc}_s(e)$ .

The proof of this theorem will be given in the Sub-section 5.1.

The third major result relates the behavioural *acceptances* of a process term with the *acceptances* of its denotation. In general these will not be identical. For example, from the definition of  $\text{Acc}_s^b$  to be given in the next sub-section, it turns out that the only acceptance sets in  $\text{Acc}_s^b(n!x.0 + m!y.0)$  are  $\{n!\}$  and  $\{m!\}$  but  $\{n!, m!\}$  is also in  $\text{Acc}_s(\llbracket n!x.0 + m!y.0 \rrbracket_s)id$ , where  $s = \{n, m\}$ . However the sets of behavioural *acceptances* and denotational *acceptances* associated with process terms will coincide up to the kernel of  $\ll$ . For two such sets  $A, B$  let  $A \cong B$  if  $A \ll B$  and  $B \ll A$ , i.e. for every  $a \in A$  there is some  $b \in B$  such that  $b \ll a$  and conversely for every  $b \in B$  there is some  $a \in A$  such that  $a \ll b$ .

**Theorem 5.4** Suppose  $x_1 \dots x_k \vdash p : P$ . Then for every  $s$  such that  $x_i \in s$ ,  $\text{Acc}^b(p) \cong \text{Acc}_s(\llbracket p \rrbracket_s id)$  where  $id \in \text{ENV}_s$  is any environment which is the identity on  $s$ .

Subsection 5.2 is devoted to the proof of this Theorem.

With these three results we can now give the proof of Theorem 5.1:

Let  $\underline{x}$  be such that  $\underline{x} \vdash p : P$ ,  $\underline{x} \vdash q : P$  and let  $s$  be such that  $x_i \in s$  for every  $i$ . Then

$$\begin{aligned}
p\rho \sqsubseteq_{\text{must}} q\rho &\iff \text{Acc}^b(p\rho) \ll \text{Acc}^b(q\rho) \text{ by Theorem 5.2} \\
&\iff \text{Acc}_s(\llbracket p\rho \rrbracket_s id) \ll \text{Acc}_s(\llbracket q\rho \rrbracket_s id) \text{ by Theorem 5.4} \\
&\iff \llbracket p\rho \rrbracket_s id \leq \llbracket q\rho \rrbracket_s id \text{ by Theorem 5.3} \\
&\iff \llbracket p \rrbracket_s \rho \leq \llbracket q \rrbracket_s \rho
\end{aligned}$$

The last line is an application of the *Substitution Lemma*:

$$\llbracket p\rho \rrbracket_s id = \llbracket p \rrbracket_s (id \circ \rho).$$

## 5.1 Internal Full Abstraction

In this section we give the details of the internal full abstraction result for the model  $\mathbf{P}$ . In order to associate *acceptances* with elements of the domains  $\mathbf{P}_s$  we first need to develop some notation for describing their various components.

As outlined in Section 2 for any functor  $\mathbf{H}$  the predomain  $(\mathbf{N} \otimes^\top \mathbf{H})_s$  can be represented as  $(s \multimap \mathbf{H}_s)$ . Moreover if  $\sigma: \mathbf{H} \multimap \mathbf{K}$  then the action of the natural transformation  $(id \otimes^\top \sigma): \mathbf{N} \otimes^\top \mathbf{K} \multimap \mathbf{N} \otimes^\top \mathbf{K}$  at  $s$  is given, in terms of this representation, by the morphism  $(\mathbf{N} \otimes^\top \mathbf{H})_s: (s \multimap \mathbf{H}) \rightarrow (s \multimap \mathbf{K})$ , which can be described by  $\lambda h. \sigma_s \circ h$ .

A representation for the domain  $(\mathbf{N} \Rightarrow \mathbf{H})_s$  was also briefly touched upon in Section 2. This takes the form of the collection of functions  $\mathcal{F}_s(\mathbf{H})$  with domain  $\mathcal{N}$ . We can also describe the actions of natural transformations in terms of these representations. If  $\sigma: \mathbf{H} \multimap \mathbf{K}$  then the action of the natural transformation  $(id \Rightarrow \sigma): (\mathbf{N} \Rightarrow \mathbf{H}) \multimap (\mathbf{N} \Rightarrow \mathbf{K})$  at  $s$  is given by the morphism  $(id \Rightarrow \sigma)_s: \mathcal{F}_s(\mathbf{H}) \rightarrow \mathcal{F}_s(\mathbf{K})$ , described by  $\lambda f. \lambda n. (\sigma_{s \cup \{n\}} \circ f(n))$ .

The predomain  $\mathbf{C}$  is isomorphic to  $(\mathbf{N} \otimes \mathbf{P}) + \mathbf{s}(\mathbf{P})$  and it is also convenient to develop a concrete representation for the predomains  $\mathbf{C}_s$ . We know that  $(\mathbf{N} \otimes \mathbf{H})_s$  can be represented by  $(s \multimap^{ne} \mathbf{H}_s)$  and  $\mathbf{s}(\mathbf{H})_s$  is determined by an element of  $\mathbf{H}_{s \cup \{y\}}$  for an arbitrary  $y$  not in  $s$ . Combining both of these we obtain a representation of  $((\mathbf{N} \otimes \mathbf{H}) + \mathbf{s}(\mathbf{H}))_s$  as a partial function with domain  $\mathcal{N}$ . Let  $\mathcal{PF}_s(\mathbf{H})$  be the set of all *non-empty* partial functions  $f$  with domain a subset of  $\mathcal{N}$  which satisfies

- $f(n) \in \mathbf{H}_{s \cup \{n\}}$
- $domain(f) - s$  is either  $\emptyset$  or  $\mathcal{N} - s$
- for all  $n, m \in domain(f) - s$ ,  $f(n) = \mathbf{H}_{(id + m \mapsto n)} f(m)$ .

Functions in  $\mathcal{PF}_s(\mathbf{H})$  can be ordered in the standard fashion, for partial functions:  $f \leq g$  if

- $domain(g) \leq domain(f)$
- for all  $n \in domain(g)$ ,  $f(n) \leq g(n)$ .

Under this ordering  $\mathcal{PF}_s(\mathbf{H})$  is a predomain isomorphic to  $((\mathbf{N} \otimes \mathbf{H}) + \mathbf{s}(\mathbf{H}))_s$ . Moreover the actions of natural transformations can also be described in terms of this representation. Let  $\sigma: \mathbf{H} \multimap \mathbf{K}$ . Then the action of the natural transformation

$$(id \otimes \sigma + \mathbf{s}(\sigma)): (\mathbf{N} \otimes \mathbf{H} + \mathbf{s}(\mathbf{H})) \multimap (\mathbf{N} \otimes \mathbf{K} + \mathbf{s}(\mathbf{K}))$$

at  $s$  is given by the morphism  $(id \otimes \sigma + \mathbf{s}(\sigma))_s: \mathcal{PF}_s(\mathbf{H}) \rightarrow \mathcal{PF}_s(\mathbf{K})$  described by  $\lambda f. \lambda n \in domain(f). (\sigma_{s \cup \{n\}} \circ f(n))$ .

These representations will be useful in reasoning about elements of  $\mathbf{P}$ . In particular they can be used to develop a convenient notation for elements of  $\mathbf{P}_s$ . Let  $d$  be an element of  $\mathbf{P}_s$  which is different from  $\perp$ . Then, modulo the isomorphisms *unfold* and *fold*,

1. let  $\mathcal{A}_s(d)$  denote  $(\pi_1 \circ down)d$ ;  $\mathcal{A}_s(d)$  is an acceptance set over  $s + s$ . Using the convention that  $in_l(n)$ ,  $in_r(n)$  represent  $n?$ ,  $n!$  respectively this can be taken to be an acceptance set over *Ind*.

2. let  $d^?$  denote  $(\pi_2 \circ \text{down})d$ ; using the representation given above  $d^?$  is an element of  $(s \rightarrow \mathbf{F}_s)$ .
3. let  $d^!$  denote  $(\pi_3 \circ \text{down})d$  which also can be taken to be an element of  $(s \rightarrow \mathbf{C}_s)$ .

With this notation we can now associate *acceptances* with elements of  $\mathbf{P}_s$ . For any  $d \in \mathbf{P}_s$  let

1.  $d \models_s \Omega$  if  $\text{unfold}(d) = \perp$
2.  $d \models_s \mathbf{A}$  if  $\mathbf{A} \in \text{Acc}_s(d)$
3.  $d \models_s \mathbf{n?x.a}$  if  $d^? n x \models_{s \cup \{x\}} \mathbf{a}$
4. for  $x \in s$ ,  $d \models_s \mathbf{n!x.a}$  if  $d^! n x \models_s \mathbf{a}$
5.  $d \models_s \nu(\mathbf{y}).\mathbf{n!y.a}$  if for some  $z \notin s$ ,  $d^! n z \models_{s \cup \{z\}} \mathbf{a}[z/y]$

The uniformity of the family of domains  $\mathbf{P}_s$  means that satisfiability of acceptances can be transferred from one to the other. Any  $s \xrightarrow{\sigma} s'$  can also be viewed as a substitution over names; it leaves untouched all those which do not appear in its domain  $s$ . So, viewing acceptances as simple terms, this means that these injections can also be applied to acceptances. With this notation we have

**Proposition 5.5** *If  $s \xrightarrow{\sigma} s'$  then  $d \models_s \mathbf{a}$  implies  $\mathbf{P}_\sigma(d) \models_{s'} \mathbf{a}\sigma$ .*

**Proof:** By induction on the structure of  $\mathbf{a}$ , by analysing the action of  $\mathbf{P}_\sigma$  on the representations of the components of  $\mathbf{P}$  given above.  $\square$

The most significant properties of these acceptances are given in the following lemma:

**Lemma 5.6**

1.  $d \models_s \mathbf{n?x.a}$  if and only if  $d^? n x \models_{s \cup \{x\}} \mathbf{a}$
2. for  $x \in s$ ,  $d \models_s \mathbf{n!x.a}$  if and only if  $d^! n x \models_s \mathbf{a}$
3.  $d \models_s \nu(\mathbf{y}).\mathbf{n!y.a}$  if and only if for every  $m \notin s$ ,  $d^! n m \models_{s \cup \{m\}} \mathbf{a}[m/y]$

**Proof:** The first two statements follow directly from the definitions. The third follows from the previous Proposition, using the uniformity of the functions in  $\mathcal{PF}_s(\mathbf{P})$ ; in particular the fact that  $f(n) = \mathbf{P}_{(s+m \mapsto n)}f(m)$ .  $\square$

Let  $\text{Acc}_s(d)$  denote the set  $\{\mathbf{a} \mid d \models_s \mathbf{a}\}$ .

**Proposition 5.7** *For all  $d, e \in \mathbf{P}_s$ ,  $d \leq e$  implies  $\text{Acc}_s(e) \ll \text{Acc}_s(d)$ .*

**Proof:** It is straightforward to show by induction on  $\mathbf{a}$  that  $\forall s \forall d, e \in \mathbf{P}_s, d \leq e, e \models_s \mathbf{a}$  implies  $d \models_s \mathbf{a}'$  for some  $\mathbf{a}' \ll \mathbf{a}$ .  $\square$

The converse however is less straightforward as it depends on the fact that  $\mathbf{P}$  is the initial solution to its defining equation. We refer the reader to [9] for the general underlying theory but here we simply state the relevant characteristics of  $\mathbf{P}$ . We define three families of natural transformations

$$\begin{aligned} \mathbf{pid}^k: \mathbf{P} &\rightarrow \mathbf{P} \\ \mathbf{fid}^k: \mathbf{F} &\rightarrow \mathbf{F} \\ \mathbf{cid}^k: \mathbf{C} &\rightarrow \mathbf{C} \end{aligned}$$

as follows:

1.  $\mathbf{aid}^0 = \perp$ , i.e. for  $\mathbf{A} = \mathbf{P}, \mathbf{F}, \mathbf{C}$   $\mathbf{aid}^0$  is the natural transformation whose action at every  $s$  is given by the constant morphism  $\lambda x. \perp$
2.  $\mathbf{fid}^{k+1} = (id \Rightarrow \mathbf{pid}^k)$
3.  $\mathbf{cid}^{k+1} = (id \otimes \mathbf{pid}^k) + \mathbf{s}(\mathbf{pid}^k)$
4.  $\mathbf{pid}^{k+1} = (id \times (id \otimes^\top \mathbf{fid}^{k+1}) \times (id \otimes^\top \mathbf{cid}^{k+1}))_\perp$

We state without proof:

**Theorem 5.8** For  $\mathbf{A} = \mathbf{P}, \mathbf{C}, \mathbf{F}$ ,  $\bigvee_k (\mathbf{aid}^k) = id_{\mathbf{A}}$ .  $\square$

With this characterisation of the domains we can now prove

**Proposition 5.9** For all  $d, e \in \mathbf{P}_s$ ,  $\text{Acc}_s(e) \ll \text{Acc}_s(d)$  implies  $d \leq e$ .

**Proof:** From the previous Theorem it is sufficient to show that for all  $k$ ,  $\mathbf{pid}_s^k(d) \leq \mathbf{pid}_s^k(e)$ . The case  $k = 0$  is trivial and we prove the case  $k + 1$  under the assumption that it is true for  $k$ .

We may assume  $d \neq \perp$  and since  $\text{Acc}(e) \ll \text{Acc}(d)$  this also means that  $e \neq \perp$ . Using the representations developed above this means that  $\text{down}(\mathbf{pid}_s^{k+1}(d))$  and  $\text{down}(\mathbf{pid}_s^{k+1}(e))$  may be taken to be

$$\mathcal{A}_s(d) \times (\mathbf{fid}_s^{k+1} \circ d^?) \times (\mathbf{cid}_s^{k+1} \circ d^!) \text{ and } \mathcal{A}_s(e) \times (\mathbf{fid}_s^{k+1} \circ e^?) \times (\mathbf{cid}_s^{k+1} \circ e^!)$$

respectively. Now  $\text{Acc}_s(e) \ll \text{Acc}_s(d)$  means that  $\cup \mathcal{A}_s(e) \subseteq \cup \mathcal{A}_s(d)$  and therefore  $\mathcal{A}_s(e) \subseteq \mathcal{A}_s(d)$ , since the latter are acceptance sets. It remains to prove

$$\begin{aligned} \mathbf{fid}_s^{k+1} \circ d^? &\leq \mathbf{fid}_s^{k+1} \circ e^? \\ \mathbf{cid}_s^{k+1} \circ d^! &\leq \mathbf{cid}_s^{k+1} \circ e^! \end{aligned}$$

As an example we prove the latter. Both  $\mathbf{cid}_s^{k+1} \circ d^!$  and  $\mathbf{cid}_s^{k+1} \circ e^!$  are partial functions, using the representations above, and so we first must demonstrate that the domain of  $\mathbf{cid}_s^{k+1} \circ e^!$  is contained in that of  $\mathbf{cid}_s^{k+1} \circ d^!$ . However this follows immediately from the fact that  $\text{Acc}_s(e) \ll \text{Acc}_s(d)$  since  $n \in \text{domain}(e^!)$  if and only if  $e \models_s n! \mathbf{x}. \mathbf{a}$  for some  $\mathbf{x}, \mathbf{a}$ .

So suppose  $n \in \text{domain}(e!)$ . We must show that

$$(\mathbf{cid}_s^{k+1} \circ d!)n \leq (\mathbf{cid}_s^{k+1} \circ e!)n$$

as objects in  $\mathbf{C}_s$ , i.e.  $\mathcal{PF}_s(\mathbf{P})$ . Using the representation of the action of functors described above  $(\mathbf{cid}_s^{k+1} \circ d!)n$ ,  $(\mathbf{cid}_s^{k+1} \circ e!)n$  work out to be

$$\lambda m \in \text{domain}(d!n).\mathbf{pid}^k(d!nm) \text{ and } \lambda m \in \text{domain}(e!n).\mathbf{pid}^k(e!nm)$$

respectively. So we must prove

- $\text{domain}(e!n) \subseteq \text{domain}(d!n)$ .

For any  $m \in s$ ,  $m \in \text{domain}(e!n)$  if and only if  $e \models_s \mathbf{n!m.a}$  for some  $\mathbf{a}$ . For any  $m \notin s$ ,  $m \in \text{domain}(e!n)$  if and only if  $e \models_s \nu(\mathbf{z}).\mathbf{n!z.a}$  for some  $\mathbf{a}$ . Therefore the required property follows from  $\text{Acc}(e) \ll \text{Acc}(d)$ .

- for every  $m \in \text{domain}(e!n)$ ,  $\mathbf{pid}^k(d!nm) \leq \mathbf{pid}^k(e!nm)$ .

First suppose  $m \in s$ . Then using part 2 of Lemma 5.6 we have that  $\text{Acc}_s(e) \ll \text{Acc}_s(d)$  implies  $\text{Acc}_s(e!nm) \ll \text{Acc}_s(d!nm)$ . So we can apply induction to obtain the required result.

If  $m \notin s$  then, using the third part of the same Lemma, we have  $\text{Acc}_s(e!nm) \ll \text{Acc}_s(d!nm)$ , since  $e \models_s \nu(\mathbf{z}).\mathbf{n!z.a}$  if and only if  $e!nm \models_{s \cup \{m\}} \mathbf{a}[m/z]$ , and once more it follows by induction that  $\mathbf{pid}^k(d!nm) \leq \mathbf{pid}^k(e!nm)$ .

□

Combining these two propositions we get Theorem 5.3:

**Theorem 5.10** Internal Full Abstraction. *For every  $d, e \in \mathbf{P}_s$ ,  $d \leq e$  if and only if  $\text{Acc}_s(e) \ll \text{Acc}_s(d)$ .* □

## 5.2 Relating Behavioural and Denotational *acceptances*

The central technical device in this section is that of *head normal form*,  $hnf$ . We show that, using equations which are sound for both the behavioural preorder and the order induced by the denotational model on process terms, every convergent term can be reduced to a  $hnf$ . The structure of these  $hnfs$  are such that it is relatively straightforward to relate their behavioural and denotational *acceptances*.

The set of basic inequations are given in Figure 3, where for convenience we add a new constant to the language,  $\Omega$ , to denote  $\mathbf{rec} X.X$ . These include the defining equations for *choice domains* together with many standard equations associated with the  $\pi$ -calculus. A typical example of the latter is:

$$\nu(n).m? \lambda y.X = m? \lambda y.\nu(n).X \text{ if } n \neq y, n \neq m$$

Others are more commonly associated with testing style equivalences and typical examples of these include:

$$\begin{aligned} n? \lambda x.X + n? \lambda x.Y &= n? \lambda x.X \oplus n? \lambda x.Y \\ n!x.X + n!x'.Y &= n!x.X \oplus n!x'.Y \end{aligned}$$

Some care must be taken when applying these to process terms. In particular they can not be applied under input prefixes as the inference rule

$$\frac{p \leq q}{n? \lambda x. p \leq n? \lambda x. q}$$

is not in general sound for  $\sqsubseteq_{must}$ . For example let  $p, q$  denote

$$m? \lambda z. x!w. 0 + x!w. m? \lambda z. 0, \quad (m? \lambda z. 0) \parallel (x!w. 0)$$

respectively. Then  $p \sqsubseteq_{must} q$  but  $n? \lambda x. p \not\sqsubseteq_{must} n? \lambda x. q$  because  $n? \lambda x. q$  can perform the action  $n?m$  followed by a communication to be transformed into 0 whereas  $n? \lambda x. p$  does not have this possibility. So  $n? \lambda x. p$  must satisfy the test  $n!m. m? \lambda x. \omega$  whereas  $n? \lambda x. q$  may fail it. Note that this rule is also not sound for the denotational semantics, i.e. if  $\llbracket p \rrbracket \leq \llbracket q \rrbracket$  then  $\llbracket n? \lambda x. p \rrbracket \not\leq \llbracket n? \lambda x. q \rrbracket$ . However we are trying to relate the behaviour of a process term  $p$ , as captured by  $\text{Acc}^b(p)$ , with a particular component of its denotation  $\llbracket p \rrbracket$ , namely  $\text{Acc}_s(\llbracket p \rrbracket, id)$ .

The proof system we use for manipulating terms is given in Figure 5 and the only difference from the standard equational proof system is the more complicated Input rule; to infer  $n? \lambda x. p \leq n? \lambda x. q$  we must establish  $p[z/x] \leq q[z/x]$  for *every* name  $z$  (although it will well-known, [21], that it is possible to restrict this to a finite number). We write  $\vdash p \leq q$  if we can derive  $p \leq q$  in the proof system using the basic equations and the interleaving law in Fig 4. It is straightforward to establish

**Proposition 5.11**  $\vdash p \leq q$  implies

1.  $p \ll q$
2.  $\llbracket p \rrbracket_s id \leq \llbracket q \rrbracket_s id$  for any  $id \in \text{ENV}_s$  which is the identity on  $s$ . □

Note that in general  $\vdash p \leq q$  does not mean  $\llbracket p \rrbracket_s \sigma \leq \llbracket q \rrbracket_s \sigma$  for every  $\sigma$ . In particular the interleaving law is not sound when interpreted in this manner. For example let  $p, q$  be the processes defined above,

$$m? \lambda z. x!w. 0 + x!w. m? \lambda z. 0, \quad (m? \lambda z. 0) \parallel (x!w. 0),$$

and let  $\sigma(m) = \sigma(n) = k$ . Then  $\vdash p \leq q$  but  $\llbracket p \rrbracket_s \sigma \not\leq \llbracket q \rrbracket_s \sigma$ .

We now explain the syntax of *hnfs*.

**Definition 2** Let  $\mathcal{A}$  be an acceptance set over *Ind*. For each  $c \in \cup \mathcal{A}$  let  $p_c$  be a process term given as follows:

- If  $c$  is  $n?$  then  $p_c$  has the form  $n? p^{n?}$  where  $p^{n?}$  is some abstraction term
- $c$  has the form  $n!$  then  $p_c$  has the form

$$n! y_1 . p^{(n!y_1)} + \dots + n! y_k . p^{(n!y_k)} \{ + \nu(y) . n! y . p^{(\nu, y)} \}$$

where  $p^{(n!y_i)}, p^{(\nu, y)}$  are arbitrary process terms,  $y_i, y$  are distinct names and  $\{ + \dots \}$  indicates an optional term.



$$\begin{aligned}
X \oplus X &= X \\
X \oplus Y &= Y + X \\
(X \oplus Y) \oplus Z &= X \oplus (Y \oplus Z) \\
X \oplus Y &\leq X \\
X + X &= X \\
X + Y &= Y + X \\
(X + Y) + Z &= X + (Y + Z) \\
X + 0 &= X \\
X + (Y \oplus Z) &= (X + Y) \oplus (X + Z) \\
X \oplus (Y + Z) &= (X \oplus Y) + (X \oplus Z) \\
\\
n ? \lambda x. X + n ? \lambda x. Y &= n ? \lambda x. (X \oplus Y) \\
n ! x. X + n ! x. Y &= n ! x. (X \oplus Y) \\
n ? \lambda x. X + n ? \lambda x. Y &= n ? \lambda x. X \oplus n ? \lambda x. Y \\
n ! x. X + n ! x'. Y &= n ! x. X \oplus n ! x'. Y \\
\\
\nu(n). (X + Y) &= \nu(n). X + \nu(n). Y \\
\nu(n). (X \oplus Y) &= \nu(n). X \oplus \nu(n). Y \\
\nu(n). n ? X &= 0 \\
\nu(n). n ! x. X &= 0 \\
\nu(n). m ? \lambda y. X &= m ? \lambda y. \nu(n). X \text{ if } n \neq y, n \neq m \\
\nu(n). m ! y. X &= m ! y. \nu(n). X \text{ if } n \neq y, n \neq m \\
\nu(n). \nu(m). X &= \nu(m). \nu(n). X \\
\\
(X \oplus Y) \parallel Z &= (X \parallel Z) \oplus (Y \parallel Z) \\
X \parallel Y &= Y \parallel X \\
\\
X + \Omega &\leq \Omega \\
X \parallel (Y + \Omega) &= \Omega \\
\Omega &\leq X \\
\nu(n). \Omega &= \Omega \\
\\
\text{rec } X. t &= t[\text{rec } X. t / X]
\end{aligned}$$

Figure 3: Equations

---

Let  $p, q$  denote  $\sum\{pre_i p_i \mid i \in I\}, \sum\{pre_j q_j \mid j \in J\}$ , where each  $pre_k$  has the form  $n!x$  or  $n?\lambda y.$ , and assume that every bound variable is different from the free variables in  $p, q$ . Then

$$p \parallel q = \begin{cases} ext(p, q) & \text{if } comms(p, q) = false \\ (ext(p, q) + int(p, q)) \oplus int(p, q) & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} ext(p, q) &= \sum\{pre_i.(p_i \parallel q, i \in I\} + \sum\{pre_j.(p \parallel q_j), j \in J\} \\ int(p, q) &= \sum\{r_{i,j}, comm(i, j)\} \end{aligned}$$

where  $comms(p, q)$  is true if there is some  $i, j$  such that  $comm(i, j)$  and  $comm(i, j)$  holds when

1.  $pre_i$  is  $n!x$  and  $pre_j$  is  $n?\lambda y.$ ; then  $r_{i,j}$  is  $p_i \parallel (q_j[x/y])$
2.  $pre_i$  is  $\nu(x).n!x$  and  $pre_j$  is  $n?\lambda y.$ ; then  $r_{i,j}$  is  $(w)(p_i[w/x] \parallel q_j[w/y])$  where  $w$  is not free in  $\nu(x).p_i$  or in  $\nu(y).q_j$
3.  $pre_i$  is  $n?\lambda y.$  and  $pre_j$  is  $n!x$ ; then  $r_{i,j}$  is  $p_i[x/y] \parallel q_j$
4.  $pre_i$  is  $n?\lambda y.$  and  $pre_j$  is  $\nu(x).n!$ ; then  $r_{i,j}$  is  $(w)(p_i[w/y] \parallel q_j[w/x])$  where  $w$  is not free in  $\nu(y).p_i$  or in  $\nu(x).q_j$

Figure 4: Interleaving Law

---

Then

$$\sum_{A \in \mathcal{A}} \left\{ \sum_{c \in A} \{p_c\} \right\}$$

is a *hnf*.

**Proposition 5.12** *If  $p \Downarrow$  then there exists a hnf,  $hnf(p)$  such that  $\vdash p = hnf(p)$ .*

**Proof:** It is virtually identical to that of Proposition 4.3 of [12] and is therefore omitted. The new operator  $\nu(n).$  is accommodated by the third group of equations in Figure 3; if  $h$  is a *hnf* then these can be used to reduce  $\nu(n).h$  to a *hnf*. The only new ingredient in the proof is to ensure that in the subterms  $h_{n!}$  there is at most one summand of the form  $\nu(x).n!x.q$ . If during the reduction procedure two such summands are generated then they can be replaced by one using the equations as follows:

$$\begin{aligned} \nu(x).n!x.q + \nu(y).n!y.q' &= \nu(z).n!z.(q[z/x]) + \nu(z).n!z.(q'[z/y]) \\ &\quad \text{by } \alpha\text{-conversion, where } z \text{ is new} \\ &= \nu(z).(n!z.q[z/x] + n!z.q'[z/y]) \\ &= \nu(z).n!z.(q[z/x] + q'[z/y]) \end{aligned}$$

□

*Hnfs* play a crucial role in

---

I	$\frac{}{p \leq p}$	$\frac{p \leq q, q \leq r}{p \leq r}$
II	$\frac{p_i \leq q_i}{op(p_1 \dots p_n) \leq op(q_1 \dots q_n)}$ <p style="text-align: center;"><math>op = +, \oplus, \parallel, \nu(n).XS</math></p>	
Eq	$\frac{}{p \leq q}$ <p>for every instance of an inequation</p>	
$\alpha$	$\frac{p \equiv_\alpha q}{p = q}$	
Input	$\frac{p[z/x] \leq q[z/x] \text{ for all names } z}{n ? \lambda x. p \leq n ? \lambda x. q}$	
If	$\frac{p \leq q}{\text{if } b \text{ then } p \text{ else } p' \leq \text{if } b \text{ then } q \text{ else } q'}$	$\frac{\llbracket b \rrbracket = \text{true}}{\text{if } b \text{ then } X \text{ else } Y = X}$
	$\frac{p' \leq q'}{\text{if } b \text{ then } p \text{ else } p' \leq \text{if } b \text{ then } q \text{ else } q'}$	$\frac{\llbracket b \rrbracket = \text{false}}{\text{if } b \text{ then } X \text{ else } Y = Y}$

Figure 5: The Proof System

---

**Lemma 5.13** *If  $\underline{x} \vdash h : P$ ,  $s$  contains all  $x_i$  and  $id$  is any environment in  $ENV_s$  which is the identity on  $s$  then  $p \Downarrow$  if and only if  $\llbracket p \rrbracket_{s, id} \neq \perp$ .*

**Proof:** (*Outline*) First suppose  $p \Downarrow$ . In this case we know that  $p$  has a *hnf*  $h$  and an examination of the definitions of  $+_{\mathbf{P}}$ ,  $\oplus_{\mathbf{P}}$  shows that  $\llbracket p \rrbracket_{s, id}$  is different from  $\perp$ .

The converse depends on the remark that for any *recursion free* process term  $d$ , if  $d \Uparrow$  then  $\llbracket d \rrbracket_{s, \perp} = \perp$ . This can be proven as follows: one can show by structural induction on  $d$  that  $d \Uparrow$  implies  $\vdash d = \Omega$  and since  $\llbracket \Omega \rrbracket = \perp$  and the proof system is sound for the interpretation  $\llbracket \cdot \rrbracket_{s, id}$  this means  $\llbracket d \rrbracket_{s, id} = \perp$ .

Now suppose  $\llbracket p \rrbracket_{s, id} \neq \perp$ . Let  $p^n$ ,  $n \geq 0$ , be the infinite sequence of *recursion free* terms obtained by unrolling occurrences of sub-terms of the form  $\text{rec } \mathbf{X}. t$ . Then standard techniques from denotational semantics can be used to show that  $\llbracket p \rrbracket = \bigvee_n \llbracket p^n \rrbracket$  which in turn means that for some  $k$   $\llbracket p^k \rrbracket_{s, id}$  is different from  $\perp$ . From the above remark we therefore have that  $p^k \Downarrow$ . Now  $p^k \leq p$  can be derived in the proof system and since this

is sound with respect to  $\ll$  it follows that  $p \Downarrow$ .  $\square$

The syntactic structure of *hnfs* ensure that we can determine the various components of their denotations. This is the topic of the next three Lemmas where we assume  $h$  is a *hnf* of the form described in Definition 2 above, such that  $\underline{x} \vdash h : P$  and  $x_i \in s$  for every  $x_i$  in  $\underline{x}$ . For any  $\sigma \in ENV_s$ ,  $\llbracket h \rrbracket_s \sigma \in \mathbf{P}_s$  and we will see how the components of  $\llbracket h \rrbracket_s \sigma$  are determined by the syntax of  $h$ . For simplicity we assume that  $\sigma$  is an injection.

**Lemma 5.14**  $\text{Acc}_s(\llbracket h \rrbracket_s \sigma) = \sigma(\mathcal{A})$

**Proof:** An examination of the definition of  $+_{\mathbf{P}}$  and  $\oplus_{\mathbf{P}}$  as it applies to *hnfs*.  $\square$

As an object in  $\mathbf{P}_s$ ,  $\llbracket h \rrbracket_s \sigma$  is different from  $\perp$  and therefore using the notation developed in Section 5.1  $(\llbracket h \rrbracket_s \sigma)^?$  can be considered as an element in  $s \rightarrow \mathbf{F}_s$ . For any  $m? \in \cup \mathcal{A}$ ,  $h^{m?}$  is an abstraction term and therefore  $\llbracket h^{m?} \rrbracket_s \sigma \in \mathbf{F}_s$ . If  $\sigma(m) = n$  then

**Lemma 5.15**  $(\llbracket h \rrbracket_s \sigma)^? n = \llbracket h^{m?} \rrbracket_s \sigma$

**Proof:** Again a simple examination of the interpretation of *hnfs*.  $\square$

In a similar manner  $(\llbracket h \rrbracket_s \sigma)^!$  can be considered as an element of  $s \rightarrow \mathbf{C}_s$  and  $(\llbracket h \rrbracket_s \sigma)^! n$ , a function in  $\mathcal{PF}_s(\mathbf{P}_s)$ , can be defined using  $h^{m!}$ . Suppose this has the form

$$m!y_1 . h_1 + \dots + m!y_k . h_k + \nu(y) . m!y . h_y.$$

Let  $sem_{m!}(h) \in \mathcal{PF}(\mathbf{P}_s)$  be defined by

$$sem_{m!}(h)z = \begin{cases} \llbracket h_i \rrbracket_s \sigma & z = \sigma(y_i) \\ \llbracket h_y \rrbracket_{s \cup \{z\}} \sigma[z/y] & z \notin s \end{cases}$$

Again an examination of the interpretation of *hnfs* shows that

**Lemma 5.16**  $(\llbracket h \rrbracket_s \sigma)^! n = sem_{m!}(h)$ .  $\square$

These lemmas make straightforward the proof of the main Theorem of this section:

**Theorem 5.17** (Theorem 5.4). *Suppose  $x_1 \dots x_k \vdash p : P$ . Then for every  $s$  such that  $x_i \in s$ ,  $\text{Acc}^b(p) \cong \text{Acc}_s(\llbracket p \rrbracket_s id)$  where  $id \in ENV_s$  is any environment which is the identity on  $s$ .*

**Proof:** We prove by structural induction on  $\mathbf{a}$  that

1.  $p \models^b \mathbf{a}$  implies  $\llbracket p \rrbracket_s id \models_s \mathbf{a}'$  for some  $\mathbf{a}'$  such that  $\mathbf{a}' \ll \mathbf{a}$
2.  $\llbracket p \rrbracket_s id \models_s \mathbf{a}$  implies  $p \models^b \mathbf{a}'$  for some  $\mathbf{a}'$  such that  $\mathbf{a}' \ll \mathbf{a}$ .

If  $\mathbf{a} = \Omega, \mathbf{A}$  then this follows from Lemma 5.13, Lemma 5.14 respectively. We consider two other cases, when  $\mathbf{a}$  has the form  $n?y.b$ ,  $\nu(m) . n!m.b$  respectively, and in both we may assume that  $p$  is a *hnf*.

1. Suppose  $p \models^b n?y.b$ . Let  $p^{n?}$  have the form  $\lambda x.t$ . This means that  $t[y/x] \models^b b$ . Applying induction we have

$$\llbracket t[y/x] \rrbracket_{s \cup \{y\}} id \models_{s \cup \{y\}} \mathbf{b}'$$

for some  $\mathbf{b}'$  such that  $\mathbf{b}' \ll \mathbf{b}$ . By the *Substitution Lemma* this means

$$\llbracket t \rrbracket_{s \cup \{y\}} (id[x \mapsto y]) \models_{s \cup \{y\}} \mathbf{b}'.$$

However if we calculate  $(\llbracket p \rrbracket_s id)^{n?}$ , as an element of  $\mathcal{F}(\mathbf{P}_s)$ , using Lemma 5.15, we obtain the function

$$\lambda k \in \mathcal{N}. \llbracket t \rrbracket_{s \cup \{k\}} (id[x \mapsto k])$$

and so by definition  $\llbracket p \rrbracket_s id \models_s n?y.\mathbf{b}'$ .

All of these steps are reversible and therefore the converse also holds.

2. Suppose  $\llbracket p \rrbracket_s id \models_s \nu(\mathbf{m}).n!\mathbf{m}.b$  and for convenience let us assume that  $m \in s$ . Using Lemma 5.16 this means that for all  $z \notin s$

$$sem_{n!}(p)z \models_{s \cup \{z\}} b[z/\mathbf{m}]$$

In particular choosing  $z$  to be  $m$  we have

$$\llbracket p_y \rrbracket_{s \cup \{m\}} (id[y \mapsto m]) \models_{s \cup \{m\}} b.$$

Using the *Substitution Lemma* this means

$$\llbracket p_y[m/y] \rrbracket_{s \cup \{m\}} id \models_{s \cup \{m\}} b.$$

By induction there exists some  $\mathbf{b}' \ll \mathbf{b}$  such that  $p_y[m/y] \models^b \mathbf{b}'$  and since  $p \xrightarrow{\nu(\mathbf{m}).n!\mathbf{m}} p_y[m/y]$  it follows that  $p \models \nu(\mathbf{m}).n!\mathbf{m}.\mathbf{b}'$ .

Again these steps are reversible and thus the converse also holds.

□

We have used the proof system developed here only to reduce process terms to head normal forms. However it is also possible to show that it is complete with respect to the model, for recursion free terms. Specifically we can show, for any recursion free process term  $d$  and any process term  $q$ , that  $\vdash d \leq q$  if and only  $\llbracket d \rrbracket_s id \leq \llbracket q \rrbracket_s id$ , where  $s$  is any set including the free names occurring in  $d, q$ .

### 5.3 Behavioural Characterisation

The behavioural characterisation of process terms using *acceptances* depends on a detailed analysis of the behavioural semantics. This was originally given in terms of a reduction relation whose definition uses a structural equivalence and consequently it is often not straightforward to derive its properties. However the following can be proved by proof induction on the definition of  $\xrightarrow{\tau}$ :

**Lemma 5.18**  $p \xrightarrow{\tau} q$  implies  $p\sigma \xrightarrow{\tau} q\sigma$  for every substitution  $\sigma$ .

The following lemma will assist with the behavioural analysis.

**Lemma 5.19**  $p \parallel q \xrightarrow{\tau} r$  if and only if one of the following:

1.  $r \equiv p' \parallel q$  where  $p \xrightarrow{\tau} p'$
2.  $r \equiv p \parallel q'$  where  $q \xrightarrow{\tau} q'$
3.  $r \equiv p' \parallel q'$  where  $p \xrightarrow{n!y} p'$  and  $q \xrightarrow{n!y} q'$
4.  $r \equiv (\nu(z) . p' \parallel q')$  for all  $z \notin fv(p, q)$  and

$$\begin{aligned} & p \xrightarrow{x!z} p', \quad q \xrightarrow{\nu(z) . n!z} q' \\ \text{or} \quad & q \xrightarrow{x!z} q', \quad p \xrightarrow{\nu(z) . n!z} p' \end{aligned}$$

□

For each acceptance set  $\mathbf{a}$  we design a specific test  $t(\mathbf{a})$  with the property that

$$\forall \mathbf{a}' \ll \mathbf{a}. p \not\models^b \mathbf{a}' \text{ implies } p \text{ must } t(\mathbf{a}).$$

The definition of these tests actually requires two extra parameters, one a finite set of names, the other a finite subset of  $Ind$ .

1.  $\mathbf{a} = \Omega$

$$t(\mathbf{a})_X^B = 1.\omega \text{ (an abbreviation for } \omega \oplus \omega)$$

2.  $\mathbf{a} = A$

$$t(\mathbf{a})_X^B = \sum \{ n!y . \omega \mid n! \in B \} + \sum \{ n? \lambda y . \omega \mid n? \in B \}$$

3.  $\mathbf{a} = n?y.b$

$$t(\mathbf{a})_X^B = 1.\omega + n!y . t(\mathbf{b}_{X \cup \{y\}}^B)$$

4.  $\mathbf{a} = n!y.b$

$$t(\mathbf{a})_X^B = 1.\omega + n? \lambda z . \text{if } z = y \text{ then } t(\mathbf{b}_X^B \text{ else } \omega), \text{ where } z \notin fv(t(\mathbf{b}_{X \cup \{y\}}^B), y))$$

5.  $\mathbf{a} = \nu(y) . n!y.b$

$$t(\mathbf{a})_X^B = 1.\omega + n? \lambda z . \text{if } z \in X \text{ then } w \text{ else } t(\mathbf{b}_{X \cup \{y\}}^B), \text{ where } z \notin fv(t(\mathbf{b}_{X \cup \{y\}}^B), y) \text{ and } z \in X \text{ is an abbreviation for } z = x_1 \vee \dots \vee x_k.$$

We leave the reader to show that these tests satisfy the following properties:

**Lemma 5.20** For every  $\mathbf{a}$

1.  $p \text{ must } t(\mathbf{a})_X^B \text{ implies that } p \text{ must } t(\mathbf{a})_{X'}^{B'}$  for every  $B' \supseteq B$

2. for every injective substitution  $\sigma$ ,  $(t(\mathbf{a})_X^B)\sigma = t(\mathbf{a}\sigma)_{X\sigma}^{B\sigma}$ .

□

The choice of the parameters to these tests,  $X$  and  $B$ , will depend on the free names occurring in the process term under consideration, and the set of *indications* associated with the *acceptance* being examined. The latter is defined by

- $A(\Omega) = \emptyset$
- $A(B) = B$
- $A(\alpha.b) = A(b)$

**Proposition 5.21** *If  $fv(p) \subseteq X$  then  $(\forall a'$  such that  $a' \ll a, p \not\models^b a')$  if and only if there exists some  $B$  such that  $B \cap A(\mathbf{a}) = \emptyset$  and  $p$  must  $t(\mathbf{a})_X^B$ .*

**Proof:** For convenience let us write  $p\checkmark$  to indicate that for every computation

$$p = p_0 \xrightarrow{\tau} p_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} p_k \xrightarrow{\tau} \dots$$

there is some  $k$  such that  $p_k \xrightarrow{\omega}$ . Then  $p$  must  $e$  if and only if  $(p \parallel e)\checkmark$ . We also let  $Ls(\mathbf{a})$  denote the set  $\{\mathbf{a}' \mid \mathbf{a} \ll \mathbf{a}'\}$ .

The proof is by induction on  $\mathbf{a}$ . We examine four cases.

- $\mathbf{a} = \Omega$

$p \models^b \mathbf{a}'$  for no  $\mathbf{a}' \in Ls(\mathbf{a})$  if and only if  $p \Downarrow$  if and only if  $(p \parallel 1.\omega)\checkmark$  if and only if  $p$  must  $t(\mathbf{a})_X^B$  for all  $B$ .

- $\mathbf{a} = A$

First suppose that  $p \models^b \mathbf{a}'$  for no  $\mathbf{a}' \in Ls(\mathbf{a})$ . This means  $p \not\models^b \Omega$ , i.e.  $p \Downarrow$ , and also that whenever  $p \xrightarrow{\tau}^* q$  then there is some indication  $i_q$  in  $S(q) - A$ . The required  $B$  is  $\{i_q \mid p \xrightarrow{\tau}^* q\}$ .

Conversely suppose  $p \parallel t(\mathbf{a})_X^B$  for some  $B$  such that  $B \cap A(\mathbf{a}) = \emptyset$ . This means that

- $p \Downarrow$ , i.e.  $p \not\models^b \Omega$
- for any  $q$  such that  $p \xrightarrow{\tau}^* q$ ,  $S(q) \cap B \neq \emptyset$ ; this in turn means that  $p \models A'$  for no  $A' \subseteq A$

Combining these we obtain that  $p \not\models^b \mathbf{a}'$  for no  $\mathbf{a}' \in Ls(\mathbf{a})$ .

- $\mathbf{a} = n?y.b$

First suppose that  $p \not\models^b \mathbf{a}'$  for every  $\mathbf{a}' \in Ls(\mathbf{a})$ . Again we know that  $p \Downarrow$ . Let  $D_{n?y}$  be the finite set  $\{p' \mid p \xrightarrow{n?y} p'\}$ . for each  $p' \in D_{n?y}$  we know that if  $\mathbf{b}' \ll \mathbf{b}$  then  $p' \not\models^b \mathbf{b}'$  and therefore by induction some finite set  $B_{p'}$  exists such that  $(p' \parallel t(\mathbf{b}')_{X \cup \{y\}}^{B_{p'}})\checkmark$ . The required  $B$  is then  $\bigcup \{B_{p'} \mid p' \in D_{n?y}\}$ . Note that when  $D_{n?y}$  is empty  $B$  is also empty.

Conversely suppose that  $(p \parallel t(\mathbf{a})_X^B)\checkmark$  for some  $B$ . Suppose  $\mathbf{a}' \ll \mathbf{a}$ . We must show that  $p \not\models^b \mathbf{a}'$ . This is straightforward if  $\mathbf{a}'$  is  $\Omega$  since  $(p \parallel t(\mathbf{a})_X^B)\checkmark$  guarantees that  $p \Downarrow$ . So we can assume that  $\mathbf{a}'$  has the form  $n?y.b'$  where  $\mathbf{b}' \ll \mathbf{b}$ . In this case it is sufficient to show that  $p' \not\models^b \mathbf{b}'$  whenever  $p \xrightarrow{n?y} p'$ . However this is guaranteed by induction since  $(p \parallel t(\mathbf{a})_X^B) \xrightarrow{\tau}^* p' \parallel t(\mathbf{b}')_{X \cup \{y\}}^B$

- $\mathbf{a} = \nu(y).n?y.b$

First suppose that  $p \not\equiv^b \mathbf{a}'$  whenever  $\mathbf{a}' \ll \mathbf{a}$ . If  $p \xrightarrow{\nu(z).n!z} p'$  for no  $z$ ,  $p$  then the required  $B$  is the empty set.

So suppose  $p \xrightarrow{\nu(z).n!z} p'$  where  $z \notin fv(\mathbf{a})$ . Then  $p' \not\equiv^b \mathbf{b}'$  for every  $\mathbf{b}'$  such that  $\mathbf{b}'[y/z]$ . For if  $p' \equiv^b \mathbf{b}'$  for such a  $\mathbf{b}'$  then  $p \equiv^b \nu(z).n!z.\mathbf{b}'$  which contradicts the hypothesis since  $\nu(z).n!z.\mathbf{b}' \ll \nu(y).n!y.\mathbf{b}$ .

So we can apply induction to find some  $B$  such that  $(p' \parallel t(b)_{X \cup \{z\}}^B)\checkmark$ . We show that  $(p \parallel t(a)_X^{B[y/z]})\checkmark$  for this choice of  $B$ .

The only significant move from this term is a communication via a bound output to  $\nu(m).p'[m/y] \parallel ((t(\mathbf{b}_{X \cup \{y\}}^{B[y/z]})[m/y])$  Using the first part of Lemma 5.20 this is the same as  $\nu(m).p'[m/y] \parallel t(\mathbf{b}[m/z]_{X \cup \{m\}}^{B[m/z]})$ . Further Lemma 5.18 can be used to show that  $q\checkmark$  implies  $(q\sigma)\checkmark$  whenever  $\sigma$  is an injective substitution since  $(q\sigma)\sigma^{-1}$  is the same as  $q$ , up to  $\alpha$ -conversion. Since  $[m/z]$  is an injective substitution it follows from  $(p' \parallel t(b)_{X \cup \{z\}}^B)\checkmark$  that  $(p' \parallel t(b)_{X \cup \{z\}}^B)[m/z]\checkmark$ , i.e.  $(p'[m/y] \parallel t(\mathbf{b}[m/z]_{X \cup \{m\}}^{B[m/z]}))\checkmark$  and therefore  $(\nu(m).p'[m/y] \parallel t(\mathbf{b}[m/z]_{X \cup \{m\}}^{B[m/z]}))\checkmark$

The converse is very similar and is left to the reader.

□

**Proposition 5.22** *If  $p \ll q$  then  $p \sqsubseteq_{must} q$ .*

**Proof:** The proof has the same structure in [10], Lemma 4.4.13, but the details are more complicated because of the different forms of communication allowed in the  $\pi$ -calculus. Suppose  $p \ll q$  and  $p \text{ must } e$ . We must show  $q \text{ must } e$  by proving that for every computation

$$e \parallel q = r_0 \xrightarrow{\tau} \dots \xrightarrow{\tau} r_k \xrightarrow{\tau} \dots \quad (*)$$

there exists some  $i$  such that  $r_i \xrightarrow{\omega}$ .

First suppose that this computation is finite, i.e.  $r_k \not\xrightarrow{\tau}$  for some  $k$ . Each  $r_i$  has the form  $\nu(\underline{v}_i).r_i$  where the individual names in  $\underline{v}$  arise because of the possible exchange of private names between the two processes. By concentrating on the interaction between the two processes, and using Lemma 5.19 this computation can be decomposed into two derivations giving the individual contributions from  $e$ ,  $q$  respectively:

$$e = e_0 \xrightarrow{\bar{a}_1} \dots e_j \dots \xrightarrow{\bar{a}_k} e_m$$

and

$$q = q_0 \xrightarrow{a_1} \dots q_j \dots \xrightarrow{a_k} q_m$$

These are such that for each  $j$  there is some  $i$  such that  $r_i = e_j \parallel q_j$  and the pair  $a_i, \bar{a}_i$  take one of the forms

- $n?x, n!x$



- $n!x, n?x$
- $n?x, \nu(x).n!x$
- $\nu(x).n!x, n?x$

for some  $x$ . Moreover we can assume that in the last two cases the particular bound name  $x$  is chosen so that it does not occur in  $fv(p)$ . Further, because  $r_k \not\rightarrow$  we also know that  $S(q_m) \cap \overline{S(e_m)} = \emptyset$ , where  $\overline{S} = \{n? \mid n! \in S\} \cup \{n! \mid n? \in S\}$ . For convenience let  $s$  denote the sequence of actions  $a_i$ ,  $S$  the set of indications  $S(q_m)$  and  $\mathbf{a}$  the *acceptance*  $\mathfrak{sS}$ . Since  $p \ll q$  this means that  $p \models^b \mathbf{b}$  for some  $\mathbf{b} \ll \mathbf{a}$ .

There are now two cases to consider:

1.  $\mathbf{b}$  has the form  $\mathfrak{sT}$ . By definition  $p \xrightarrow{s} p'$  for some  $p'$  such that  $p' \not\rightarrow$  and  $S(p') \subseteq T \subseteq S$ . This computation, from  $p$  to  $p'$  can be zipped together with  $e \xrightarrow{\overline{s}} e_m$  to form a computation

$$e \parallel \xrightarrow{\tau} \dots \xrightarrow{\tau} e_m \parallel p'$$

The relevant features of this computation are firstly that  $e_m \parallel p' \not\rightarrow$  and secondly it only uses  $e_i$  which occur in the original computation (\*). It follows from  $p$  *must*  $e$  that  $e_i \xrightarrow{\omega}$  for some  $i$ .

2.  $\mathbf{b}$  has the form  $\mathfrak{s}'\Omega$  where  $\mathfrak{s}'$  is some subsequence of  $\mathfrak{s}$ . The approach of the previous case also works here using the computation  $p \xrightarrow{\mathfrak{s}'} p'$  where  $p \uparrow$ .

We have not touched on the case when the computation (\*) is infinite. However this can be treated in exactly the same manner as in Lemma 4.4.13 from [10]; we simply need to know that, up to  $\alpha$ -conversion, the computation tree from any process term, with respect to the reduction relation  $\xrightarrow{\tau}$ , is finite branching.  $\square$

Combining these two propositions gives the behavioural alternative characterisation, Theorem 5.2:

**Theorem 5.23**  $p \sqsubseteq_{\text{must}} q$  if and only if  $p \ll q$ .

## 6 The May Case

In this section we briefly outline another Interpretation and show that it is fully abstract with respect to the behavioural preorder  $\sqsubseteq_{\text{may}}$ . We can use essentially the same equation as in Section 4,

$$\begin{aligned} \mathbf{P} &\cong \mathbf{A} \times (\mathbf{N} \otimes^{\top} \mathbf{F}) \times (\mathbf{N} \otimes^{\top} \mathbf{C}) \\ \mathbf{F} &\cong \mathbf{N} \Rightarrow \mathbf{P} \\ \mathbf{C} &\cong (\mathbf{N} \otimes \mathbf{P}) + \mathfrak{s}(\mathbf{P}) \end{aligned}$$

but here we solve it in in the category  $u\mathcal{D}$ , i.e. the category whose objects are functors from  $I$  to  $u\mathcal{SL}$ , the category of upper semi-lattices. Note that the only difference is that we have dropped the use of the lifting functor  $(\ )_{\perp}$ ; this is because every upper choice predomain is automatically a domain and therefore it is unnecessary to add a

least element. Moreover because of the degeneracy of much of the algebraic structure of upper choice domains there is some redundancy in this equation.

Here  $\mathbf{A}$  denotes the function which at  $s$  gives the free upper choice predomain over  $s$ . This is easily shown to be the set of subsets of  $s$  ordered by set inclusion. One can also show that in the category  $u\mathcal{D}$  the action of the functor  $(\mathbf{N} \otimes^\top A)$  at  $s$ , for any  $A$ , is described by the set of partial functions from  $s$  to  $A_s$ ,  $s \multimap A_s$ , this time ordered in the more usual fashion:  $f \leq g$  if  $\text{domain}(f) \subseteq \text{domain}(g)$  and  $f(n) \leq g(n)$  for every  $n \in \text{domain}(f)$ . Thus instead of using  $\mathbf{P}$  as described above, we could use the simpler equation

$$\begin{aligned} \mathbf{Q} &\cong (\mathbf{N} \otimes^\top \mathbf{F}) \times (\mathbf{N} \otimes^\top \mathbf{C}) \\ \mathbf{F} &\cong \mathbf{N} \Rightarrow \mathbf{Q} \\ \mathbf{C} &\cong (\mathbf{N} \otimes \mathbf{Q}) + s(\mathbf{Q}) \end{aligned}$$

Note that  $\mathbf{Q}$  is not isomorphic to  $\mathbf{P}$ , as objects in  $u\mathcal{D}$ . But  $\mathbf{Q}$  can be embedded in  $\mathbf{P}$  and is sufficiently rich to support the interpretation of all terms in the language.

To interpret the language in this new domain  $\mathbf{P}$  we proceed exactly as in Section 4; we require an interpretation for each of the language constructs. However these also can be borrowed directly from Section 4; in each of the definitions we simply omit the use of the functor  $(\ )_\perp$  and its adjoint *down*. In this way we have a new interpretation  $\llbracket \Gamma \vdash t : P \rrbracket^u$  for each typing judgement and consequently for each process term a family of interpretations  $\llbracket p \rrbracket_s^u$  from  $ENV_s$  to  $\mathbf{P}_s$ .

**Theorem 6.1**  $\llbracket p \rrbracket_s^u \rho \leq \llbracket q \rrbracket_s^u \rho$  if and only if  $p\rho \sqsubseteq_{\text{may}} q\rho$ .

The proof of this theorem is very similar to that of Theorem 5.1, although considerably simpler, and here we merely outline the necessary steps.

Let  $\text{Seq}$  be the subset of  $\text{Acc}$  consisting of all those elements of the form  $s\Omega$ . The method of associating process terms with these *sequences*, both behaviourally and denotationally, is slightly different than that used in Section 5. Behaviourally the change is in the clause:

$$p \models^b \Omega \text{ for every } p$$

and the denotational change is similar:

$$d \models_s \Omega \text{ for every } d \in \mathbf{P}_s$$

The proof of the behavioural alternative characterisation, Theorem 5.2, can be mimicked to give:

**Proposition 6.2**  $p \sqsubseteq_{\text{may}} q$  if and only if  $\text{Seq}^b(p) \subseteq \text{Seq}^b(q)$ . □.

The internal full abstraction result, Theorem 5.3, can also be repeated:

**Proposition 6.3** For all  $d, e \in \mathbf{P}_s$ ,  $d \leq e$  if and only if  $\text{Seq}_s(d) \leq \text{Seq}_s(e)$ . □.

The final ingredient is the result corresponding to Theorem 5.4:

**Proposition 6.4** If  $\underline{x} \vdash p : P$ ,  $x_i \in s$  for every  $i$  and  $id \in ENV_s$  is the identity on  $s$  then  $\text{Seq}^b(p) = \text{Seq}(\llbracket p \rrbracket_s^u id)$ . □.

These three results can be combined, as in Section 5, to give a proof for Theorem 6.1.

We end this section and the paper with a brief look at how these models can be used to characterise more general forms of testing. As an illustration we use *may testing*.

Let us write  $p \Downarrow_{may}$  if  $p \xrightarrow{\tau} q^*$  for some  $q$  which can perform some communication, i.e.  $q \xrightarrow{n?m}$  or  $q \xrightarrow{n!m}$  for some  $n, m$ .

**Definition 3** *Let  $p \sqsubseteq_{mayconv} q$  if  $C[p] \Downarrow_{may}$  implies  $C[q] \Downarrow_{may}$  for every context  $C[\ ]$ .*

This is the form of testing used in, for example [5], and is a natural generalisation of that used for the  $\lambda$ -calculus in [2].

**Theorem 6.5**  *$p \sqsubseteq_{mayconv} q$  if and only if  $\llbracket p \rrbracket \leq \llbracket q \rrbracket$ .*

**Proof:** (*Outline*) First suppose  $\llbracket p \rrbracket \leq \llbracket q \rrbracket$  and  $C[p] \Downarrow_{may}$ . Then by the construction of the model  $\llbracket C[p] \rrbracket \leq \llbracket C[q] \rrbracket$ . Let  $s$  be a sufficiently large set of names which includes all those free in both  $C[p]$  and  $C[q]$ . Then we have  $\llbracket C[p] \rrbracket_s id \leq \llbracket C[q] \rrbracket_s id$  and from Proposition 6.4 this means  $\text{Seq}^b(C[p]) \subseteq \text{Seq}^b(C[q])$ . In general  $r \Downarrow_{may}$  if and only if  $\text{Seq}(r)$  is not  $\{\epsilon\}$ . and therefore it follows that  $C[q] \Downarrow_{may}$ .

Conversely suppose that for all contexts  $C[\ ]$ ,  $C[p] \Downarrow_{may}$  implies  $C[q] \Downarrow_{may}$ . We must show  $\llbracket p \rrbracket \leq \llbracket q \rrbracket$ . For this it is sufficient to show that  $\llbracket p \rrbracket_s \rho \leq \llbracket q \rrbracket_s \rho$  for every  $s$  and for every environment  $\rho$ . From the full abstraction result for *may testing* it is sufficient in turn to show that for every process  $e$  that  $p\rho \text{ may } e$  implies  $q\rho \text{ may } e$ . However it is easy to simulate *may testing* in terms of testing for  $\Downarrow_{may}$  in arbitrary contexts and substitutions can be simulated using input prefixing. For example the effect of the substitution of  $x$  for  $y$  in  $r$  can be obtained by the context  $\nu(n).(n? \lambda y. p \parallel n!x.0)$ .  $\square$

## A Definition of Parallel

The natural transformation  $\parallel_{\mathbf{P}}: \mathbf{P} \otimes \mathbf{P} \rightarrow \mathbf{P}$  is defined to be  $\mathbf{par}^{\otimes}$ , where  $\mathbf{par}: (\mathbf{P} \times \mathbf{P}) \rightarrow \mathbf{P}$  is a natural transformation which is linear in both its arguments, and is defined, in analogy with the interleaving law, in terms of two auxiliary natural transformations,  $\mathbf{ext}$ ,  $\mathbf{int}$ , by

$$(+_{\mathbf{P}} \circ \langle \mathbf{ext} \times \mathbf{int} \rangle) \wedge \mathbf{int}$$

These natural transformations are in turn defined in terms of  $\mathbf{par}$  and therefore in fact we take  $\mathbf{par}$  to be the least fixpoint of its definition.

To define  $\mathbf{ext}$ ,  $\mathbf{int}$  we need to develop some notation. For any  $\tau: \mathbf{P} \times \mathbf{P} \rightarrow \mathbf{P}$  let  $\tau_{\mathbf{F}}: (\mathbf{N} \Rightarrow \mathbf{P}) \times \mathbf{P} \rightarrow (\mathbf{N} \Rightarrow \mathbf{P})$  be the natural transformation defined by letting  $(\tau_{\mathbf{F}})_s: (\mathbf{N} \Rightarrow \mathbf{P})_s \times \mathbf{P}_s \rightarrow (\mathbf{N} \Rightarrow \mathbf{P})_s$  be the morphism

$$(\tau_{\mathbf{F}})_s(f, d) = \lambda x \in \mathcal{N}. \tau_{s \cup \{x\}}(f(x), d).$$

We can also define a natural transformations  $\tau_{\mathbf{C}}: \mathbf{C} \times \mathbf{P} \rightarrow \mathbf{C}$  by letting  $(\tau_{\mathbf{C}})_s(f, d)$  be the element in  $\mathbf{C}$  defined by  $\lambda x \in \text{domain}(f). \tau_{s \cup \{x\}}(f(x), d)$ . Any natural transformation

$\tau: (A \times B) \rightrightarrows C$  can be extended in an obvious way to a natural transformation in  $(\mathbf{N} \otimes^\top A \times B) \rightrightarrows \mathbf{N} \otimes^\top C$  and let us use the non-standard notation  $\mathbf{N} \otimes^\top \tau$  for this extension.

Then the natural transformation  $\mathbf{ext}: (\mathbf{P} \times \mathbf{P}) \rightrightarrows \mathbf{P}$  is given by  $(\mathbf{par}_\mathbf{A} \times \mathbf{par}_\mathbf{F} \times \mathbf{par}_\mathbf{C})_\perp$  where

- $\mathbf{par}_\mathbf{A}$  is  $+_\mathbf{A} \circ \langle \pi_1 \circ \mathit{down} \circ \pi_1 \times \pi_1 \circ \mathit{down} \circ \pi_2 \rangle$
- $\mathbf{par}_\mathbf{F}$  is  $((\mathbf{N} \otimes^\top \mathbf{par}_\mathbf{F}) \circ \langle \pi_2 \circ \mathit{down} \circ \pi_1 \times \pi_2 \rangle) \wedge ((\mathbf{N} \otimes^\top \mathbf{par}_\mathbf{F}) \circ \langle \pi_2 \circ \mathit{down} \circ \pi_2 \times \pi_1 \rangle)$
- $\mathbf{par}_\mathbf{C}$  is  $((\mathbf{N} \otimes^\top \mathbf{par}_\mathbf{C}) \circ \langle \pi_3 \circ \mathit{down} \circ \pi_1 \times \pi_2 \rangle) \wedge ((\mathbf{N} \otimes^\top \mathbf{par}_\mathbf{C}) \circ \langle \pi_3 \circ \mathit{down} \circ \pi_2 \times \pi_1 \rangle)$

Intuitively  $\mathbf{ext}_s(d, e)$  describes the element of  $\mathbf{P}_s$  obtaining by interleaving the possible actions of the elements  $d, e$  of  $\mathbf{P}_s$  without taking into consideration possible *communications* between them.

These *communications* are the concern of the second natural transformation  $\mathbf{int}$ . For any  $\tau: (\mathbf{P} \times \mathbf{P}) \rightrightarrows \mathbf{P}$  let  $\tau_{\mathbf{FC}}: (\mathbf{F} \times \mathbf{C}) \rightrightarrows \mathbf{P}$  be defined by letting  $(\tau_{\mathbf{FC}})_s: (\mathbf{F}_s \times \mathbf{C}_s) \rightarrow \mathbf{P}_s$  be the morphism defined by

$$(\tau_{\mathbf{FC}})_s(f, g) = \sum \{ \tau_{s \cup \{n\}}(f(n), g(n)) \mid n \in \mathit{domain}(g) \}$$

This is now generalised to a natural transformation

$$\tau_{\mathbf{FC}}^\mathbf{N}: (\mathbf{N} \otimes^\top \mathbf{F}) \times (\mathbf{N} \otimes^\top \mathbf{C},) \rightrightarrows \mathbf{P}^\top$$

by the definition

$$(\tau_{\mathbf{FC}}^\mathbf{N})_s(g_1, g_2) = \begin{cases} \sum \{ (\tau_{\mathbf{FC}})_s(g_1 n, g_2 n) \mid n \in s' \} & s' \neq \emptyset \\ \top & s' = \emptyset \end{cases}$$

where  $s' = \mathit{domain}(g_1) \cap \mathit{domain}(g_2)$ . Note that the codomain of this natural transformation needs to be  $\mathbf{P}^\top$  rather than  $\mathbf{P}$ .

Finally the natural transformation  $\mathbf{int}: (\mathbf{P} \times \mathbf{P}) \rightrightarrows \mathbf{P}^\top$  is defined by

$$(\tau_{\mathbf{FC}}^\mathbf{N} \circ \langle \pi_2 \circ \pi_1 \times \pi_3 \circ \pi_2 \rangle) \wedge (\tau_{\mathbf{FC}}^\mathbf{N} \circ \langle \pi_2 \circ \pi_2 \times \pi_3 \circ \pi_1 \rangle)$$

**Acknowledgements:** The author would like to thank Cosimo Laneve, Dusko Pavlovic and Davide Sangiori for useful comments on earlier drafts of this paper.

## References

- [1] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.
- [2] S. Abramsky and C. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 1989. to appear.
- [3] Henk Barendregt. *The Lambda Calculus*. North-Holland, 1984. Studies in logic 103.
- [4] M. Boreale and R. DeNicola. Testing for mobile processes. In *Proceedings of CONCUR 92*, 1992. To appear in *Information and Computation*.
- [5] G. Boudol. A lambda calculus for (strict) parallel functions. *Information and Computation*, 108:51–127, 1994.
- [6] R. Crole. *Categories for Types*. Cambridge University Press, 1993.
- [7] R. DeNicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.
- [8] F.J.Oles. Type algebras, functor categories and block structure. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, pages 543–573. Cambridge University Press, 1985.
- [9] Carl Gunter. *Semantics of Programming Languages*. MIT Press, 1992.
- [10] M. Hennessy. *An Algebraic Theory of Processes*. MIT Press, 1988.
- [11] M. Hennessy. A Model for the  $\pi$  Calculus. Technical Report 8/91, University of Sussex, 1991.
- [12] M. Hennessy and A. Ingolfsdottir. A theory of communicating processes with value-passing. *Information and Computation*, 107(2):202–236, 1993.
- [13] D.J. Howe. Equality in lazy computation systems. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science*, pages 198–203, 1989.
- [14] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
- [15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part i. *Information and Computation*, 100(1):1–40, 1992.
- [16] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part ii. *Information and Computation*, 100(1):41–77, 1992.
- [17] Robin Milner. The polyadic  $\pi$ -calculus: a tutorial. In *Proc. International Summer School on Logic and Algebra of Specification*, Marktobendorf, 1991.
- [18] Eugenio Moggi. Notions of computation and monad. *Information and Computation*, 93:55–92, 1991.

- [19] E. Moggi, M.P.Fiore and D. Sangiorgi. A Fully-Abstract Model for the  $\pi$ -Calculus. To appear in Proceedings of LICS'96, 1996.
- [20] P.W. O'Hearn and R.D. Tennant. Semantics of local variables. In *Applications of Categories in Computer Science*, pages 217–238. London Mathematical Society Lecture Note Series, 1992.
- [21] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Information and Computation*, 120:174–197, 1995.
- [22] A. M. Pitts and I. Stark. On the observable properties of higher order functions that dynamically create local names, or: What's new. In *Proc. of Mathematical Foundations of Computer Science*, pages 122–141. Springer Verlag, 1993.
- [23] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. Phd thesis, Edinburgh University, Scotland, 1992.
- [24] D. Sangiorgi. A theory of bisimulation for the  $\pi$ -calculus. In *Proceedings of the CONCUR'93*, number 715 in LNCS, 1993.
- [25] Ian Stark. A Fully-Abstract Domain Model for the  $\pi$ -Calculus. To appear in Proceedings of LICS'96, 1996.