FINDING PRESHEAF MODELS FOR THE FINITE π-CALCULUS

By
Guy Beaulieu, B.Sc.
January 2002

A Thesis
submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Science in Mathematics[1]

[1]The M.Sc. Program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

This thesis provides a fully-abstract (set theoretical) model for the finite π-calculus with respect to late-bisimulation and late-equivalence relations. This is achieved by amalgamating the works by M.P. Fiore, E. Moggi and D. Sangiorgi, and I. Stark. In their respective works the authors construct categorical models, and define a meta-language in which the finite π-calculus can be interpreted. We discuss the general properties a categorical model should satisfy to be considered an appropriate model for the finite π-calculus. In particular, I show that the categorical model based on the syntax provides a fully abstract model for the finite π-calculus. Finally, I include all the details of the model which were often omitted by the above authors. We extend the discussion by examining alternative categorical constructs for the model of the finite π-calculus, for example we use doubly closed categories which are a main focus of Bunched Logic by P.W. O'Hearn and D.J. Pym.

Acknowledgements

I would like to thank my supervisor, Philip J. Scott, for all his support and dedication to my research. His guidance was greatly appreciated, without which this thesis would comprise of one very long sentence.

Dedication

I dedicate this work to my loving wife, Daniela Gionni. Without her constant encouragement and confidence in me, I could never have gotten through the rough spots.

Contents

Chapter 1

Introduction

The π-calculus was first introduced, by Milner, Parrow and Walker [18], as a formal calculus for analyzing systems of concurrent agents or processes. Later, many interesting variants were introduced creating a much wider scope of analysis, such as the π_I-calculus [22], the fusion calculus [21], and many more. The main goal is to describe the continually changing environment or configurations of systems of interacting concurrent processes. In their most general form, the various systems of the π-calculus present a systematic approach to understanding interaction between agents based on message passing communication. From this viewpoint, the content of the message is less important than the actual sending and receiving mechanism. It is the latter which the π-calculus is trying to model. The fundamental notion needed to describe concurrent communication in the π-calculus is *naming*, i.e. labelling the links (or channels) between agents by variables (or names). The variables denote communication channels which together describe the interface of the process with its environment. Thus, two processes can communicate only if they share a channel labelled by the same variable. The continually changing dynamics results from the possibility that a process may send the name of a channel to another process as a message; thus it may create a link to the receiving process or destroy one of its own links. The dynamics of a system expressed through the π-calculus language is given by an operational semantics, i.e. the dynamic evolution of π-calculus expressions is analyzed through a relation on processes, called the *reaction relation*. This relation

describes a step-by-step evolution of the interactions a system of concurrent processes may enable. The problem here is that the syntactic analysis in such an operational semantics is very involved, described by many varying and complicated congruence relations based on variants of bisimulation. This is the traditional approach of the process algebra community [16, 18, 22, 21]

Recently, there has been a growing interest in finding a denotational approach to studying the π-calculus. That is, we are looking for ordinary mathematical models that describe the calculus. The main papers on the subject are [24, 9], both of which find a domain-theoretic model of name-passing processes based on functor category models. They have shown their models to be fully abstract; that is, the ordinary equality of (the interpretation of) processes in the model corresponds to the operational equivalence given by late-bisimulation or late-equivalence. The body of their work includes solving a straightforward recursive (pre)domain equation:

$$Proc = \mathcal{P}(N \times (N \Rightarrow Proc) + N \times N \times Proc + N \times (N \multimap Proc) + Proc) \quad (1)$$

Here $\mathcal{P}$ is an appropriate powerdomain monad due to Abramsky [1, 10]. Also, the object N, called the *object of names*, represents finite sets of free names. This equation is far from arbitrarily chosen; it recursively defines $Proc$, called the *object of processes*. This object represents the possible π-calculus actions a process can do which are represented in Equation 1 as the terms of the summation. These include input (of type: $N \times (N \Rightarrow Proc)$), output (of type $N \times N \times Proc$), bound output (of type $N \times (N \multimap Proc)$) and internal action (of type $Proc$).

We are interested in following the same approach seen in [24, 9], however we shall consider a simpler functor category which models a restricted version of the π-calculus. This simplification has been mentioned by many authors [24, 9, 11] as a possible model for the finite π-calculus, i.e. the restriction of the π-calculus to processes without replication. We develop the details here for this simplified model for the first time. Although the calculus loses some expressive power, this simplified version is still expressive enough to describe many systems. Here, instead of working with a functor category on domains, we shall be able to work with presheaf categories, i.e. functor categories defined on **Set**. Our model consists of the presheaf

category $\mathbf{Set}^{\mathbf{Inj}}$, where **Inj** is the category of finite sets and injections. In our open interpretation, processes will be represented by natural transformations from a finite product of the object of names to the object of processes. Moreover, we shall prove a full-abstraction theorem which shall state that equality in the open interpretation shall correspond to late-equivalence between the processes. On the other hand, in the closed interpretation we will have processes be elements of the object of processes. In this case, the full-abstraction theorem states that equality in the model will correspond to late-bisimulation between the processes. In both interpretations Equation 1 is interpreted using simpler categorical constructions. The construct $\mathcal{P}$ is taken to be the finite powerset monad and the object of names N is defined to be the embedding of **Inj** to **Set**. Hence our model contains many well-known categorical notions which are simpler to work with, making the restrictions on our π-calculus language understandable. Therefore, following the work done by both [24, 9], we can construct a presheaf model for the finite π-calculus.

Our work is separated in the following chapters:

Chapter 2: Relevant Category Theory

We discuss all the pertinent category theory for the development of our presheaf model. The main categorical issues to understand are the presheaf categories and their properties, the Day construction of a tensor product on presheaf categories, the construction of the free-semilattice monad, and the computation of least fixed points of functors.

Chapter 3: The Finite π-calculus

We introduce the finite π-calculus as a model of concurrent computation, with a recursive definition on its process constructors. Based mainly on [16, 17, 18], we follow the classic definition of the π-calculus and restrict it to the finite π-calculus. The important notions are the definition of the reaction relation, its relationship to labelled transition systems and the late-bisimulation and late-equivalence relations on processes.

Chapter 4: Presheaf Models of the Finite π-calculus

In this chapter, we construct our model for the finite π-calculus using the

presheaf category $\mathbf{Set}^{\mathbf{Inj}}$, where **Inj** is the category of finite sets and injections. We begin with a general discussion about the categorical properties our presheaf model should satisfy in order to be a good model. We then show that a concrete model can be constructed based on the syntax of the finite π-calculus which satisfies the general properties. Furthermore, we prove a full-abstraction theorem for two interpretations in our model. In particular, we prove how late-equivalence between finite π-calculus processes correspond to equality in the open interpretation. On the other hand, we prove how late-bisimularity between finite π-calculus processes correspond to equality in the closed interpretation.

What we have achieved in this thesis is to work out the details mentioned by [24, 9], that a functor category based on **Set** could construct a sound model for the finite π-calculus. By amalgamating much of the work from [24] and the work in [9], we have come up with a categorical approach to describing the properties a presheaf category must admit to be a valid model of the finite π-calculus. We have worked out many of the details missing in the referenced works, as well as connecting our constructions to standard **Set** based categorical presheaf semantics. In particular, we work out the details, mentioned in [24], of the use of the Day tensor construction in presheaf categories, which induces the appropriate (right)adjoint function space used in modelling the finite π-calculus.

In addition, we have made a connection between the categorical models of "Bunched Logic", from [19], and the presheaf model of the finite π-calculus. One very important similarity which we discuss here, is that the construction of both models are determined by doubly closed categories. We shall see that doubly closed categories contain many useful properties that both models exploit to their advantage.

Chapter 2

Relevant Category Theory

This chapter is concerned with some preliminary ideas for the construction of a presheaf model in Chapter 4. The main topics shall encompass:

(i) *Basic category theory.* We review pertinent topics of category theory in order to refresh the readers memory about certain properties which arise in important types of categories, such as cartesian closed, symmetric monoidal closed and doubly closed categories.

(ii) *Presheaf categories.* Our aim is to construct a presheaf model for the finite π-calculus. We shall describe how to construct presheaf categories and demonstrate some of their classical properties.

(iii) *Monads and fixed points in* $\mathbf{Set}^{\mathbf{C}}$. These categorical constructions are important in the creation of our object of processes in our presheaf model.

We begin with a discussion of the relevant category theory, which will be necessary to understand the more important categorical concepts in the following sections. We base ourselves on the main references for this subject such as MacLane [14], Borceux [5], Barr and Wells [3, 4], and Lambek and Scott [13].

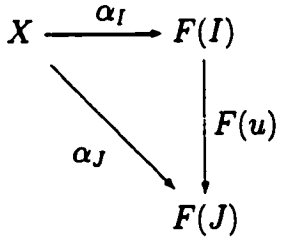
2.1 Cartesian Closed Categories

2.1.1 Limits and Colimits

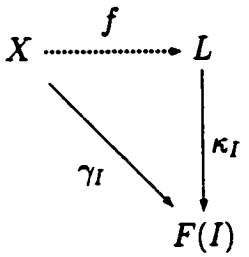
In this section we shall study limits in categories. We outline special cases of limits that are important, such as terminal objects, products, pullbacks and equalizers. We begin by describing the general construction of an arbitrary limit, as described in [14].

Definition 2.1.1 An **I**-*indexed diagram* is a functor, $F : \mathbf{I} \to \mathbf{C}$, where **I** is a small category and **C** is a locally small category.

Definition 2.1.2 Let $F : \mathbf{I} \to \mathbf{C}$ be a diagram. A *cone* α from $X \in |\mathbf{C}|$ to F (or a *cone* for the **I**-indexed diagram F in **C** with vertex X) is a collection of arrows $\langle \alpha_I : X \to F(I) \rangle_{I \in |\mathbf{I}|}$ in **C** such that for any arrow $u : I \to J$ in **I** the following diagram commutes.



Definition 2.1.3 Given a diagram $F : \mathbf{I} \to \mathbf{C}$, a *limit* for F determines and is determined by a cone $\kappa = \langle \kappa_I : L \to F(I) \rangle_{I \in |\mathbf{I}|}$ from some $L \in |\mathbf{C}|$ to F such that for any other cone $\gamma = \langle \gamma_I : X \to F(I) \rangle_{I \in |\mathbf{I}|}$ there exists a *unique* arrow $f : X \to L$ such that for every $I \in |\mathbf{I}|$ the diagram



commutes. The cone κ is called the *limiting cone.*

Fact: Due to the universal mapping property, limits (and colimits), when they exist, are unique up to isomorphism. This result is discussed in greater detail in [14].

Example 2.1.4 We give a few examples of limits as well as their interpretation in the categories **Set**, the category of sets and functions, $\mathbf{Pos}_P$, the category associated to the poset P (objects of $\mathbf{Pos}_P$ are the elements of P, and $Hom_{\mathbf{Pos}_P}(x,y) = \begin{cases} \{*\} & \text{if } x \leq y \\ \emptyset & \text{else} \end{cases}$), and **Vec**, the category of vector spaces and linear transformations.

Terminal Objects Let $\mathbf{C}$ be a category. A *terminal object* of $\mathbf{C}$, if it exists, is an object $1 \in |\mathbf{C}|$ with the universal property that for every $D \in |\mathbf{C}|$, there is a unique arrow from D to 1.

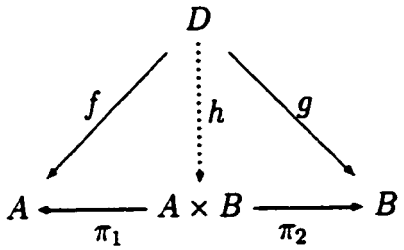
- In **Set**: The terminal object is any singleton set $\{*\} \in |\mathbf{Set}|$.
- In $\mathbf{Pos}_P$: The terminal object is the maximum element of P, when there is one.
- In **Vec**: The terminal object is the one-element vector space.

Finite Products Let $\mathbf{C}$ be a category, and $A, B \in |\mathbf{C}|$. A *product* of A and B, if it exists, is defined by:

(i) an object $A \times B \in |\mathbf{C}|$, and

(ii) a pair of arrows $\pi_1 : A \times B \to A, \pi_2 : A \times B \to B$

with the universal property that, for any object $D \in |\mathbf{C}|$ and any arrows $f : D \to A, g : D \to B$ in $\mathbf{C}$, there is a unique map $h : D \to A \times B$ such that $\pi_1 \circ h = f$ and $\pi_2 \circ h = g$, i.e. the following diagram commutes:



We denote h by $\langle f, g \rangle$ to emphasize the maps f and g which are paired.

- In **Set**: The product $A \times B = \{(a,b); a \in A, b \in B\}$, is the usual cartesian product of sets together with the projections π_1 and π_2. Here, for $d \in D$, $h(d) = (f(d), g(d)) \in A \times B$.
- In $\mathbf{Pos}_P$: The product $a \times b = inf\{a,b\}$, i.e. the infimum of a and b (if it exists). The projection maps π_1 and π_2 exist since $inf\{a,b\} \leq a$ and $inf\{a,b\} \leq b$, the universal property says: if $d \leq a, d \leq b$ then $d \leq inf\{a,b\}$. We denote $inf\{a,b\} = a \wedge b$.
- In **Vec**: The product is the usual direct sum $U \times V = \{(u,v); u \in U, v \in V\}$ with the operation $(u,v) \bullet_{U\times V} (u',v') = (u \bullet_U u', v \bullet_V v')$, and $\alpha(u,v) = (\alpha u, \alpha v)$ for $u, u' \in U, v, v' \in V$ and α a scalar. The projections $\pi_1 : U \times V \to U$ and $\pi_2 : U \times V \to V$ are as usual and are linear transformations.

Products We may now generalize our definition of products to an arbitrary family of objects. Let $\mathbf{C}$ be a category and I an indexing set. A *product* of the family of objects $\{A_i\}_{i\in I}$, if it exists, is defined by:

(i) an object $\Pi_{i\in I} A_i \in |\mathbf{C}|$, and

(ii) a family of arrows $\{\pi_j : \Pi_{i\in I} A_i \to A_j\}_{j\in I}$

with the universal property that, for any object $D \in |\mathbf{C}|$ and any family of arrows $\{\alpha_j : D \to A_j\}_{j\in I}$ in $\mathbf{C}$, there exists a unique map $h : D \to \Pi_{i\in I} A_i$ such that $\pi_j \circ h = \alpha_j$, for all $j \in I$.

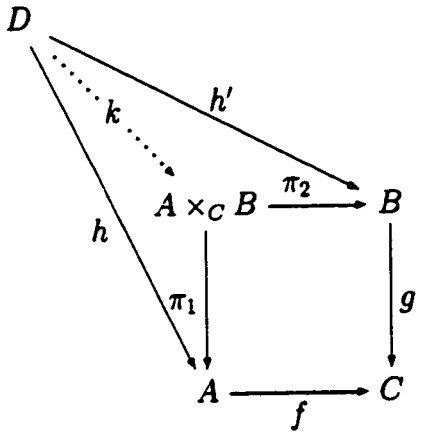
- In **Set**: The product $\Pi_{i\in I} A_i = \{(a_1, a_2, \ldots); a_i \in A_i, i \in I\}$, is the cartesian product of a family of sets, where π_i are the usual projections.
- In $\mathbf{Pos}_P$: The product on an arbitrary subset $S \subseteq P$, is denoted by $\wedge S$, is the extension of its definition on a finite subset of objects, i.e. the infimum on an infinite set of elements with $\pi_i : S \to a_i$, $a_i \in S$, given by the corresponding inequality.
- In **Vec**: The product $\sum_{i\in I} U_i = \{(u_1, u_2, \ldots); u_i \in U_i, i \in I\}$, is the extension of the finite direct sum for the category and the projections are as usual.

Pullbacks Let $\mathbf{C}$ be a category and let $f : A \to C$ and $g : B \to C$ be two arrows in $\mathbf{C}$. A *pullback* of f and g, if it exists, is defined by:

(i) an object $A \times_C B \in |\mathbf{C}|$, and

(ii) a pair of arrows $\pi_1 : A \times_C B \to A, \pi_2 : A \times_C B \to B$ such that $g \circ \pi_2 = f \circ \pi_1$

with the universal property that for any object $D \in |\mathbf{C}|$ and any arrows $h : D \to A$ and $h' : D \to B$ with $f \circ h = g \circ h'$ in $\mathbf{C}$, there is a unique arrow $k = \langle h, h' \rangle : D \to A \times_C B$ such that $\pi_1 \circ k = h$ and $\pi_2 \circ k = h'$, i.e. the following diagram commutes.



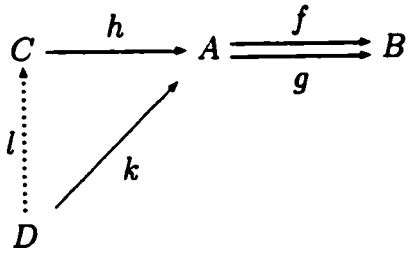
- In **Set**: Let $f : A \to C$ and $g : B \to C$ be two set functions. Let $A \times_C B = \{(a, b) \in A \times B; f(a) = g(b)\}$ and let π_1, π_2 be the restriction maps of the usual projections.
- In $\mathbf{Pos}_P$: Let $a \leq c$ and $b \leq c$ be two arrows. Let $a \times_c b = a \wedge b$, and let the corresponding maps be the inequalities $a \wedge b \leq a$ and $a \wedge b \leq b$.
- In **Vec**: Let $\lambda : U \to V$ and $\mu : U' \to V$ be two linear maps. Let $U \times_V U' = \{(u, u') \in U \times U'; \lambda(u) = \mu(u')\}$, again with π_1, π_2 the appropriately restricted projections.

Equalizers Let $\mathbf{C}$ be a category and let $f : A \to B$ and $g : A \to B$ be two arrows in $\mathbf{C}$. An *equalizer* of f and g, if it exists, is defined by:

(i) an object $C \in |\mathbf{C}|$, and

(ii) an arrow $h : C \to A$, such that $g \circ h = f \circ h$

with the universal property that for any object $D \in |\mathbf{C}|$ and any arrow $k : D \to A$ with $f \circ k = g \circ k$, there is a unique arrow $l : D \to C$ such that $h \circ l = k$, i.e. the following diagram commutes.



- In **Set**: Let $f : A \to B$ and $g : A \to B$ be two set functions. Let $C = \{a \in A; f(a) = g(a)\}$ and $h : C \to A$ be the injection of this subset into A.
- In $\mathbf{Pos}_P$: Let $a \leq b$ be an arrow in the category. Let $c = a$, then we have that $c \leq a$ and for any $d \leq a$ we get $d \leq c$. Note: equalizers always exist in this category.
- In **Vec**: Let $\lambda : U \to V$ and $\mu : U \to V$, be two linear functions. Then the equalizer Z is the vectorial subspace $Z = \{u \in U; \lambda(u) = \mu(u)\}$, together with the linear inclusion map $h : Z \to U$.

Definition 2.1.5 A category **C** is (small)-*complete* if it has all (small) limits.

However, verifying that a category has all limits can be very difficult. The following result, from [14] simplifies the work needed to check if a category is complete.

Theorem 2.1.6 *A category* **C** *is complete if and only if it has all products (indexed by any set) and equalizers of all pairs of arrows.*

Example 2.1.7

(a) **Set** and **Vec** are complete. We have shown in Example 2.1.4 that each of these categories have products and equalizers. Therefore by Theorem 2.1.6 they are complete.

(b) $\mathbf{Pos}_P$, is complete if and only if all infima exist. As we have seen in Example 2.1.4 equalizers in $\mathbf{Pos}_P$ always exist and products are infima. Therefore, by Theorem 2.1.6, $\mathbf{Pos}_P$ is complete if and only if it has all products if and only if it has all infima.

We may now dualize the concept of limits given in Definition 2.1.3, which are called *colimits*. In this way we construct a second important categorical structure from the first.

Example 2.1.8 We dualize the construction of terminal objects, products and equalizers given in Example 2.1.4 into their respective colimits. Moreover, we shall also give a specific example of each in the categories **Set**, $\mathbf{Pos}_P$, and **Vec**.

Initial Objects Let $\mathbf{C}$ be a category. An *initial object* of $\mathbf{C}$, if it exists, is an object $0 \in |\mathbf{C}|$ with the universal property that for every $D \in |\mathbf{C}|$, there is a unique arrow $0 \to D$.

- In **Set**: The initial object is the empty set.
- In $\mathbf{Pos}_P$: The initial object is the minimal element of P, when there is one.
- In **Vec**: The initial object is the trivial vector space, i.e. the vector space with only one element.

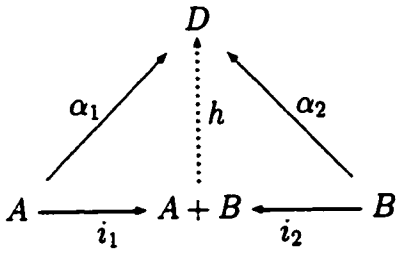
Remark 2.1.9 Notice that in **Vec**, the initial object is equal to the terminal object.

Coproducts Let $\mathbf{C}$ be a category, and $\{A_i\}_{i\in I}$, be a family of objects in $\mathbf{C}$. A *coproduct* of $\{A_i\}_{i\in I}$, if it exists, is defined by:

(i) an object $\Sigma_{i\in I}A_i \in |\mathbf{C}|$, and

(ii) a family of arrows $\{i_j : A_j \to \Sigma_{i\in I}A_i\}_{j\in I}$

with the universal property that, for any object $D \in |\mathbf{C}|$ and any family of arrows $\{\alpha_j : A_j \to D\}_{j \in I}$ in $\mathbf{C}$, there is a unique map $h = [\alpha_1, \alpha_2, \ldots] : \Sigma_{i \in I} A_i \to D$ such that $h \circ i_j = \alpha_j$, i.e. the following diagram commutes for the case of a finite family of two objects.



- In **Set**: The coproduct of A and B is the disjoint union $A + B = (A \times \{1\}) \cup (B \times \{2\})$, with the inclusions i_1 and i_2. Let $h : A + B \to D$ be defined such that $h(c, j) = i_j(c)$ for $j \in \{1, 2\}$. To extend to the infinite case, we proceed to take the infinite disjoint union.
- In $\mathbf{Pos}_P$: The coproduct of a and b is the object $a + b = sup\{a, b\} = a \vee b$, i.e. the supremum of the set $\{a, b\}$. The maps i_1 and i_2 are represented by the inequalities $a \leq a \vee b$ and $b \leq a \vee b$. We proceed to get the coproduct on an arbitrary subset $S \subseteq P$, $\vee S$, by taking the supremum of S and its related projections.
- In **Vec**: The coproduct of U and V is the direct sum of the vector spaces, i.e. $U \times V$ under the operation $(u, v) \bullet_{U \times V} (u', v') = (u \bullet_U u', v \bullet_V v')$, $\alpha(u, v) = (\alpha u, \alpha v)$, for $u, u' \in U, v, v' \in V$ and α a scalar. The inclusions are defined as follows

$$\begin{aligned} in_1 : \; U &\to U \times V \\ u &\mapsto (u, 0_V) \end{aligned}$$

and similarly for in_2. We define the unique map h such that $h(u, v) = f(u) \bullet g(v)$. We extend this construction for an arbitrary family of vector spaces.

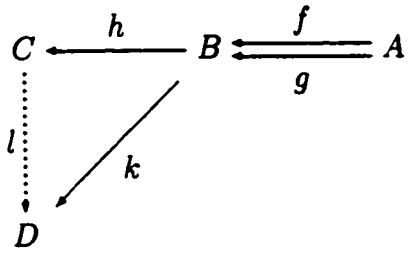
Remark 2.1.10 Notice that the product and the coproduct object for the category of **Vec** coincide. Both are represented by the direct sum of vector spaces.

Coequalizers Let $\mathbf{C}$ be a category and let $f : A \to B$ and $g : A \to B$ be two arrows in $\mathbf{C}$. A *coequalizer* of f and g, if it exists, is defined by:

(i) an object $C \in |\mathbf{C}|$, and

(ii) an arrow $h : B \to C$, such that $h \circ g = h \circ f$

with the universal property that for any object $D \in |\mathbf{C}|$ and any arrow $k : B \to D$ with $k \circ f = k \circ g$, there is a unique arrow $l : C \to D$ such that $l \circ h = k$, i.e. the following diagram commutes.



- In **Set**: Let $f, g : A \to B$ be arrows. The coequalizer of f and g is the set $B/\sim$, where $\sim \subseteq B \times B$ is the least equivalence relation which contains all pairs $(f(a), g(a))$, for $a \in A$. The arrow h is simply the quotient map.
- In $\mathbf{Pos}_P$: Let $a \leq b$ be an arrow of the category. The coequalizer is the element c, where $c = b$. Note: coequalizers always exist in $\mathbf{Pos}_P$
- In **Vec**: Let $\lambda, \mu : V \to U$ be two linear maps. The coequalizer of λ and μ is the quotient space U/Y such that $Y = \{\lambda(v) - \mu(v); v \in V\}$. The linear map h represents the quotient map between U and U/Y.

Definition 2.1.11 A category $\mathbf{C}$ is (small) *cocomplete* if it has all colimits.

We can also dualize Theorem 2.1.6 for cocomplete categories.

Theorem 2.1.12 *A category is cocomplete if and only if it has all coproducts (indexed by any set) and coequalizers of all pairs of arrows.*

Example 2.1.13

(a) **Set** and **Vec** are cocomplete by Theorem 2.1.12 with the coproducts and coequalizers in each category shown in Example 2.1.8.

(b) $\mathbf{Pos}_P$, is cocomplete if and only if it has all suprema. As we have seen in Example 2.1.8, coequalizers always exist and coproducts of elements are represented by their suprema. Thus, by Theorem 2.1.12 we have that $\mathbf{Pos}_P$ is cocomplete if and only if all coproducts exist if and only if all suprema exist.

Remark 2.1.14 Notice that $\mathbf{Pos}_P$ has all infima (i.e. is complete) if and only if it has all suprema (i.e. it is cocomplete) if and only if it is a complete lattice. Suppose that $\mathbf{Pos}_P$ has all infima. We shall show that $\mathbf{Pos}_P$ has all suprema. Let $S \subseteq P$, we claim that $\vee S = \wedge\{t \in P; t \geq S\}$, where $t \geq S$ signifies $\forall s \in S$ $t \geq s$.
We show that $\wedge\{t \in P; t \geq S\}$ is an upper bound for S, i.e. $\wedge\{t \in P; t \geq S\} \geq S$. Let $s \in S$, we have that $s \leq t, \forall t \in \{t \in P; t \geq S\} \Rightarrow$
$s \leq \wedge\{t \in P; t \geq S\} \Rightarrow S \leq \wedge\{t \in P; t \geq S\}$. Moreover, if $d \geq S$ then by the property of infima $d \geq \wedge\{t \in P; t \geq S\}$ because $d \in \{t \in P; t \geq S\}$. Therefore, we have

$$\vee S = \wedge\{t \in P; t \geq S\}$$

The converse is done similarly. Therefore, if $\mathbf{Pos}_P$ has all infima it also has all suprema. Thus, it is a complete lattice.

2.1.2 Exponentials

In this section we describe categories in which there exists a right adjoint to the product functor. These categories are fundamental to logic and theoretical computer science, as seen in [23]. We begin by reviewing the definition stated in [13] for such categories and follow with some examples.

Definition 2.1.15 A *cartesian closed category (= ccc)* is a category $\mathbf{C}$ with finite products (hence having a terminal object) such that for every $A \in |\mathbf{C}|$, the functor

$- \times A$ has a specified right adjoint, denoted $A \Rightarrow -$ called the *exponential functor* or the *function space.*

Some properties of ccc's are:

(i) If $\mathbf{C}$ is a ccc, since $(A \Rightarrow -)$ is a right adjoint to $(- \times A)$ we have the following isomorphism

$$Hom_{\mathbf{C}}(B \times A, C) \overset{\lambda}{\cong} Hom_{\mathbf{C}}(B, A \Rightarrow C)$$

Therefore, for any map $f : B \times A \to C$, there exists a corresponding map $\lambda_f : B \to A \Rightarrow C$. We call λ_f the *currying map* of f or the *exponentiation* of the map f.

(ii) There exists two important maps, the unit η and the counit *eval* such that

$$\eta_{A,B} : B \to (A \Rightarrow (B \times A)), \quad eval_{A,B} : (A \Rightarrow B) \times A \to B.$$

corresponding to $\lambda(id_{B \times A})$, and $\lambda^{-1}(id_{A \Rightarrow B})$, resp.

Moreover, both the following composites are identities.

$$(A \Rightarrow eval_{A,B}) \circ \eta_{A,A \Rightarrow B} = id_{A \Rightarrow B} \tag{2}$$

$$eval_{A,B \times A} \circ (\eta_{A,B} \times A) = id_{B \times A} \tag{3}$$

Finally, an easy theorem presented in [14], can be used to determine the existence of an adjunction between two functors.

Theorem 2.1.16 *Each adjunction is completely determined by the functors* $F : \mathbf{C} \to \mathbf{D}$ *and* $G : \mathbf{D} \to \mathbf{C}$ *and natural transformations* $\eta : I_{\mathbf{C}} \dot{\to} GF$ *and* $eval : FG \dot{\to} I_{\mathbf{D}}$ *such that both Equation 2 and Equation 3 are satisfied.*

Example 2.1.17

(a) **Set** is a ccc, with $A \Rightarrow B = Hom_{\mathbf{Set}}(A, B)$. In this case, let $eval_{A,B}(f, a) = f(a)$ and $\eta_{A,B}(a)(b) = \langle a, b \rangle$.

(b) G-**Set** the category of G-sets, i.e. a group homomorphism $h : G \to Sym(X)$, where $Sym(X)$ is the group of bijections $X \to X$, or equivalently, a set X with a mapping $\cdot : G \times X \to X$, defined by $(g, x) \mapsto g \cdot x$, called the *action*, satisfying $g_1 \cdot (g_2 \cdot x) = (g_1 g_2) \cdot x$ and $e \cdot x = x$. An arrow between G-sets, $\phi : (X, \cdot_1) \to (Y, \cdot_2)$, is a mapping $\phi : X \to Y$ such that $\phi(g \cdot_1 x) = g \cdot_2 \phi(x)$ for all $g \in G$ and $x \in X$.

The product $X \times Y$ is given by the usual cartesian product. The action map $G \times (X \times Y) \to X \times Y$ is defined by $g \cdot (x, y) = (g \cdot x, g \cdot y)$.

The exponential $X \Rightarrow Y$ is given by the set of *all* functions from X to Y. Then the action map $G \times (X \Rightarrow Y) \to X \Rightarrow Y$ is defined by $g \cdot f : X \to Y$, where $(g \cdot f)(a) = g \cdot (f(g^{-1} \cdot a))$.

To check that G-**Set** is a ccc, we need to check that

$$\begin{array}{rccc} eval_{X,Y} : & (X \Rightarrow Y) \times X & \to & Y \\ & (f, x) & \mapsto & f(x) \end{array} \quad \text{and} \quad \begin{array}{rccc} \eta_{X,Y} : & Y & \to & X \Rightarrow (Y \times X) \\ & y & \mapsto & (x \mapsto (y, x)) \end{array}$$

are G-maps and the following equations hold for any objects $X, Y \in |\mathbf{Set}|$:

$$(X \Rightarrow eval_{X,Y}) \circ \eta_{X, X \Rightarrow Y} = id_{X \Rightarrow Y}, \text{ and } eval_{X, Y \times X} \circ (\eta_{X,Y} \times X) = id_{Y \times X}$$

We begin by showing they are G-maps.

$$\begin{aligned} g \cdot (eval_{X,Y}(f, x)) &= g \cdot f(x) \\ &= g \cdot f((g^{-1} g) \cdot x) \\ &= (g \cdot f)(g \cdot x) \\ &= eval_{X,Y}((g \cdot f), g \cdot x) \\ &= eval_{X,Y}(g \cdot (f, x)). \end{aligned}$$

Similarly,

$$g \cdot \eta_{X,Y}(y) : X \to Y \times X,$$

satisfies

$$((g \cdot \eta_{X,Y}(y))(x) = g \cdot (\eta_{X,Y}(y)(g^{-1} \cdot x))$$

$$\begin{aligned} &= g \cdot (y, g^{-1} \cdot x) \\ &= (g \cdot y, x) \\ &= (\eta_{X,Y}(g \cdot y))(x). \end{aligned}$$

So $eval_{X,Y}$ and $\eta_{X,Y}$ are G-maps. To show we have that the equalities hold we check that:

$$\begin{aligned} (((X \Rightarrow eval_{X,Y}) \circ \eta_{X,X\Rightarrow Y})(f))(x) &= eval_{X,Y}((\eta_{X,X\Rightarrow Y}(f))(x)) \\ &= eval_{X,Y}((f,x)) \\ &= f(x) \\ &= (id_{X\Rightarrow Y}(f))(x). \end{aligned}$$

$$\begin{aligned} (eval_{X,Y\times X} \circ (\eta_{X,Y} \times X))(y,x) &= eval_{X,Y\times X}(\eta_{X,Y}(y), x) \\ &= (\eta_{X,Y}(y))(x) \\ &= (y,x) \\ &= id_{Y\times X}(y,x). \end{aligned}$$

Hence G-**Set** is a ccc.

(c) Another example of a ccc, from [13], is the category of complete partial orders, **Cpo**. An object $C \in |\mathbf{Cpo}|$, is a poset in which every countable ascending chain $a_0 \leq a_1 \leq a_2 \leq \ldots$ of elements has a supremum. Morphisms in the category **Cpo** are mappings which preserve suprema of countable ascending chains, (such mappings will necessarily preserve order). The product structure is inherited from **Set** and for $A, B \in |\mathbf{Cpo}|$, $A \Rightarrow B$ is $Hom_{\mathbf{Cpo}}(A,B)$ with order and suprema being defined componentwise.

2.2 Monoidal Categories

Monoidal categories are categories equipped with an abstract tensor product. These form an increasingly important class of categories with many applications in algebra,

logic and theoretical computer science, as seen in [23]. We study this structure, since it will be important in our presheaf category models of the finite π-calculus.

2.2.1 Tensor

Definition 2.2.1 A category $\mathbf{C}$ is *monoidal* (or *tensored*) if it is equipped with a functor $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$, an object $I \in |\mathbf{C}|$ and isomorphisms $A \otimes I \overset{\phi_A}{\cong} A \overset{\lambda_A}{\cong} I \otimes A$ and $A \otimes (B \otimes C) \overset{\alpha_{A,B,C}}{\cong} (A \otimes B) \otimes C$ satisfying the MacLane equations.

Definition 2.2.2 The *MacLane equations* are: Let $A, B, C, D \in |\mathbf{C}|$.

1. The isomorphism $\alpha = \alpha_{A,B,C} : A \otimes (B \otimes C) \cong (A \otimes B) \otimes C$, satisfies the MacLane Pentagon:

$$\begin{array}{ccccc} A \otimes (B \otimes (C \otimes D)) & \xrightarrow{\alpha_{A,B,C\otimes D}} & (A \otimes B) \otimes (C \otimes D) & \xrightarrow{\alpha_{A\otimes B,C,D}} & ((A \otimes B) \otimes C) \otimes D \\ \downarrow{\scriptstyle 1 \otimes \alpha_{B,C,D}} & & & & \uparrow{\scriptstyle \alpha_{A,B,C} \otimes 1} \\ A \otimes ((B \otimes C) \otimes D) & & \xrightarrow{\qquad\qquad\alpha_{A,B\otimes C,D}\qquad\qquad} & & (A \otimes (B \otimes C)) \otimes D \end{array}$$

2. The isomorphisms $\lambda_C : I \otimes C \cong C$ and $\phi_A : A \otimes I \cong A$, satisfy:

$$\begin{array}{ccc} A \otimes (I \otimes C) & \xrightarrow{\alpha_{A,I,C}} & (A \otimes I) \otimes C \\ \downarrow{\scriptstyle 1 \otimes \lambda_C} & & \downarrow{\scriptstyle \phi_A \otimes 1} \\ A \otimes C & = & A \otimes C \end{array}$$

Moreover $\lambda_I = \phi_I : I \otimes I \cong I$.

Example 2.2.3

(a) Any finite product category, where the tensor is the product.

(b) Any finite coproduct category, where the tensor is the coproduct.

(c) $\mathbf{Rel}_\times$, the category of sets and relations. Here, composition is given by the composition of relations, i.e. if $R : A \to B$ and $S : B \to C$ are relations, then define their composition $R;S = \{(a,c); \exists b \in B \; aRb \text{ and } bSc\}$. The tensor product is given by $\otimes$ = product of sets.

(d) $\mathbf{Rel}_+$, the category of sets and relations, with $\otimes = +$ the coproduct.

(e) Any commutative monoid or commutative group, viewed as a category with one object.

(f) $\mathbf{Funct}(\mathbf{C},\mathbf{C})$, the functor category whose objects are functors from $\mathbf{C}$ to $\mathbf{C}$ and arrows are natural transformations between them, where composition of functors defines the tensor product.

Remark 2.2.4 Unlike products and coproducts, notice how the monoidal structure on a category is not unique. There may exist more than one tensor product which satisfies the stated equations in a specific category. This is why we must be careful when stating that a category is monoidal to state exactly under which tensor product we are analyzing it.

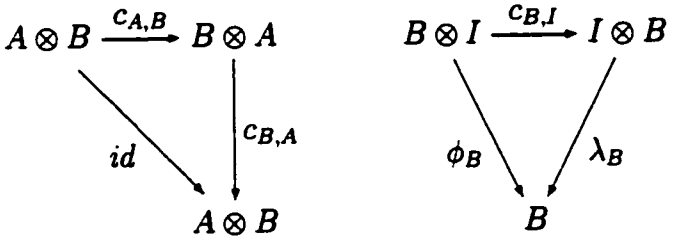
2.2.2 Symmetry

Generally, in a monoidal category, it is not dictated that the isomorphism $A \otimes B \cong B \otimes A$ must hold. However, if the tensor happens to be a product or coproduct then the above isomorphism automatically holds.

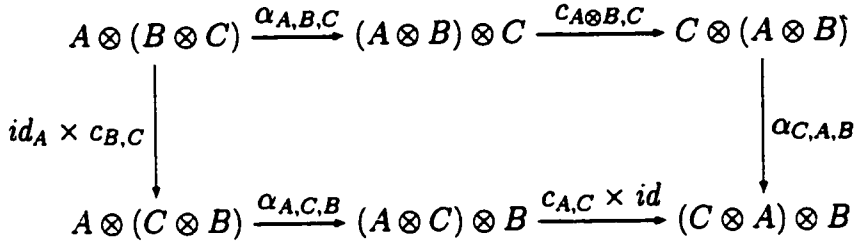
Definition 2.2.5 A monoidal category $\mathbf{C}$ is a *symmetric monoidal category (= smc)*, if for any two objects $A, B \in |\mathbf{C}|$, it is equipped with a natural transformation

$$c_{A,B} : A \otimes B \to B \otimes A$$

such that the following diagrams commute:



and



Example 2.2.6

(a) The first four examples of Examples 2.2.3 are symmetric, while the last two are not.

(b) **Inj**, the category of finite sets with injections, is a symmetric monoidal category. Take the disjoint union, denoted by $+$, to be the tensor product. It is obviously an associative and symmetric operation. For the identity object of the tensor pick the empty set, i.e. $I = \emptyset$, which satisfies the unity equations. Therefore, $(\mathbf{Inj}, +, \emptyset)$ is a smc.

2.2.3 Monoidal Closed Categories

Similarly to a ccc structure, a monoidal closed structure arises when the tensor functor $- \otimes A$ has a corresponding right adjoint $A \multimap -$. This is the second structure we shall need on our presheaf models.

Definition 2.2.7 A *monoidal closed category (=mcc)* $(\mathbf{C}, \otimes, I, \multimap)$ is a monoidal category such that for each object $A \in |\mathbf{C}|$, the functor $- \otimes A : \mathbf{C} \to \mathbf{C}$ has a specified right adjoint $A \multimap -$, (i.e. for each A there is an isomorphism, natural in B,C):

$$Hom_{\mathbf{C}}(C \otimes A, B) \overset{\lambda^\bullet}{\cong} Hom_{\mathbf{C}}(C, A \multimap B)$$

Moreover, we say that $\mathbf{C}$ is a *symmetric monoidal closed category (= smcc)* if $\mathbf{C}$ is a monoidal closed category with a symmetric tensor.

As a consequence, just as in ccc's, in any smcc there are *unit* and *counit* maps $eval^* : (A \multimap B) \otimes A \to B$ and $\eta^* : C \to (A \multimap (C \otimes A))$ determined by the adjointness. We shall try to keep close to our ccc notation. In particular there is the analog of currying, called *linear currying*, arising from the isomorphism above; it is denoted by $\lambda^*_f : C \to (A \multimap B)$ for any map $f : (C \otimes A) \to B$.

Example 2.2.8 The category $\mathbf{Vec_{fin}}$, the category of finite dimensional vector spaces and linear transformations, is a smcc. Let $U, V \in |\mathbf{Vec_{fin}}|$.

(i) The tensor is defined to be

$$U \otimes V = Hom_{\mathbf{Vec_{fin}}}(U^*, V)$$

where U^* represents the dual vector space to U, i.e. $U^* = Hom_{\mathbf{Vec_{fin}}}(U, K)$, where K is the scalar field on which U is defined.

(ii) The linear function space is defined to be

$$U \multimap V = Hom_{\mathbf{Vec_{fin}}}(U, V)$$

(iii) I is the 1-dimensional vector space.

2.3 Doubly Closed Categories

To be able to model the finite π-calculus, we shall need a category with sufficient structure. The category will need to admit a cartesian closed structure, a symmetric monoidal closed structure (which does not necessarily coincide with the cartesian closed structure), and admit finite coproducts. Such a category, called a *bicartesian doubly closed category*, defined in [19] in order to described bunched logic, will be sufficient for our purposes.

Definition 2.3.1 A *doubly closed category (= dcc)*, is a category equipped with two monoidal closed structures. A dcc is called *cartesian* if one of the closed structures is cartesian and the other symmetric monoidal and *bicartesian* if, in addition, it has finite coproducts.

In a dcc, say $\mathbf{C}$, we shall have two adjunctions due to the two closed structures in the category, one with the cartesian structure $(\mathbf{C}, \times, 1, \Rightarrow)$ and one with the symmetric monoidal structure, $(\mathbf{C}, \otimes, I, \multimap)$. Let $F, G, H \in |\mathbf{C}|$. We have the isomorphisms:

$$Hom_{\mathbf{C}}(G \times F, H) \cong Hom_{\mathbf{C}}(G, F \Rightarrow H) \quad Hom_{\mathbf{C}}(G \otimes F, H) \cong Hom_{\mathbf{C}}(G, F \multimap H)$$

In addition, the above adjointness relations give an isomorphism between the global points of $F \Rightarrow H$ and the global (monoidal) points of $F \multimap H$, in the sense that;

$$Hom_{\mathbf{C}}(1, F \Rightarrow H) \cong Hom_{\mathbf{C}}(F, H) \cong Hom_{\mathbf{C}}(I, F \multimap H)$$

An example, from [19], of a dcc, will help understand the double structure.

Example 2.3.2 **Set** $\times$ **Set** is a bicartesian closed, with coproducts and cartesian closed structures defined pointwise from their corresponding versions in **Set**. A symmetric monoidal closed structure is given by:

$$\begin{aligned} I &= (1,0), \text{ where } 1 = \{z\}, 0 = \emptyset \\ (E_0, E_1) \otimes (F_0, F_1) &= ((E_0 \times F_0) + (E_1 \times F_1), (E_0 \times F_1) + (E_1 \times F_0)) \\ (E_0, E_1) \multimap (F_0, F_1) &= ((E_0 \Rightarrow F_0) \times (E_1 \Rightarrow F_1), (E_0 \Rightarrow F_1) \times (E_1 \Rightarrow F_0)) \end{aligned}$$

Remark 2.3.3 The above example does not convey any particularly useful computational ideas but we can use it to make a few remarks.

(i) It is a non-degenerate model, in that I is not a terminal object and $\otimes$ is not cartesian product. So the definition of dcc's does not somehow induce a collapse of the specified structure.

(ii) There are no maps in the model from 1 to I. In the model we have $1 = (1,1)$ and $I = (1,0)$, thus there are no maps of the form $(1,1) \to (1,0)$, since there are no maps from 1 to 0.

(iii) $(0,1) \otimes (0,1) = (1,0)$ and $(0,1) \times (0,1) = (0,1)$. This, combined with the fact that there are no maps between $(0,1)$ and $(1,0)$ in either direction, implies that in general, there are no maps between $A \otimes B$ and $A \times B$, where $A, B \in \mathbf{Set} \times \mathbf{Set}$.

(iv) $((0,1) \Rightarrow (1,0)) \cong (0 \Rightarrow 1, 1 \Rightarrow 0) \cong (1,0)$ and $((0,1) \multimap (1,0)) =$ $((0 \Rightarrow 1) \times (1 \Rightarrow 0), (0 \Rightarrow 0) \times (1 \Rightarrow 1)) \cong (1 \times 0, 1 \times 1) \cong (0,1)$. This, combined with the fact that there are no maps between $(0,1)$ and $(1,0)$ in either direction, implies that there are no maps from $((0,1) \Rightarrow (1,0))$ to $((0,1) \multimap (1,0))$ or back.

2.4 Presheaves

In this section we will define the type of category we will use to construct a model for the finite π-calculus. It is a category which admits all the properties we have discussed in this chapter and hence will have a lot of categorical structure to exploit when constructing our model. We begin by defining the category and by showing that it is cartesian closed.

2.4.1 Presheaf Functors

First, we describe the objects in a presheaf category.

Definition 2.4.1 Let $\mathbf{C}$ be a fixed small category. A *presheaf* is a contravariant functor from $\mathbf{C}^{op}$ to the category **Set**, i.e., it is a covariant functor $P : \mathbf{C} \rightarrow \mathbf{Set}$.

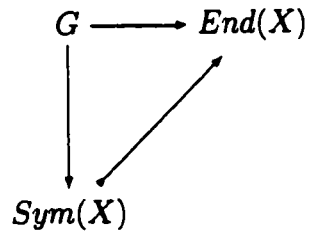
Example 2.4.2 Let G be a group. Define a category $\mathbf{C}_G$ as follows: $|\mathbf{C}_G| = \{*\}$ and $Hom_{\mathbf{C}_G}(*,*) = G$, with composition the multiplication in G, and identity the element e_G, the unit of G. Therefore for each set X a presheaf P_X exists, $P_X : \mathbf{C}_G \rightarrow \mathbf{Set}$ such that:

(i) on the object $* \in |\mathbf{C}_G|$, $P_X(*) = X$.

(ii) On arrows $f \in Hom_{\mathbf{C}_G}(*,*)$,

$$\begin{array}{rccl} P_X f : & P_X * & \rightarrow & P_X * \\ & x & \mapsto & (P_X f)(x), \text{ which we denote by } f \cdot x \end{array}$$

Since P_X is a functor the following is true: for $f, h \in G$ we have the arrows $f : * \to *$ and $h : * \to *$. Since $P_X(fh) = P_X(f) \circ P_X(h)$, this means for all $x \in X$, $fh \cdot x = f \cdot (h \cdot x)$. Similarly, $P_X(e_G) = id_{P_X(*)}$, so for all $x \in X$, $e_G \cdot x = x$. So, $P_X(-) : G \to End(X)$ determines a monoid homomorphism. Since G is actually a group (and since monoid homomorphisms are automatically group homomorphisms if G is a group), we have a factorization:



Hence we can consider presheaves $P_X : \mathbf{C}_G \to \mathbf{Set}$ as G-sets. We shall discuss natural transformations between these presheaves in Example 2.4.7.

2.4.2 Yoneda Lemma

We have defined a presheaf to be a covariant functor from $\mathbf{C}$ to $\mathbf{Set}$. Among them there is a special functor $h^A \equiv Hom_{\mathbf{C}}(A, -)$ which sends the object $A' \in |\mathbf{C}|$ into the set $Hom_{\mathbf{C}}(A, A')$ and the arrow $f : A' \to A''$ onto the mapping $Hom_{\mathbf{C}}(1_A, f) : Hom_{\mathbf{C}}(A, A') \to Hom_{\mathbf{C}}(A, A'')$, defined by for $g : A \to A'$, $Hom_{\mathbf{C}}(1_a, f)(g) = f \circ g$. Which brings us to an important result called Yoneda's Lemma.

Lemma 2.4.3 (Yoneda Lemma) *If $F : \mathbf{C} \to \mathbf{Set}$ is a functor and $C \in |\mathbf{C}|$ (for $\mathbf{C}$ a small category), there is a bijection*

$$y : Nat(h^C, F) \cong FC$$

which sends each natural transformation $\alpha : h^C \stackrel{\cdot}{\to} F$ to $\alpha_C(id_C)$, the image of the identity $id_C : C \to C$.

Definition 2.4.4 A functor $F : \mathbf{C} \to \mathbf{D}$ is *faithful* if the induced mapping $\phi : Hom_{\mathbf{C}}(C, C') \to Hom_{\mathbf{D}}(FC, FC')$ which sends $f : C \to C'$ in $\mathbf{C}$ to

$Ff : FC \to FC'$ for all $C, C' \in |\mathbf{C}|$ is injective and we say the functor is *full* if the same map is surjective. Moreover, we say that the functor is a *full embedding* if it is full and faithful. Note that if the functor F is contravariant the induced mapping above becomes $\phi : Hom_{\mathbf{C}}(C, C') \to Hom_{\mathbf{D}}(FC', FC)$.

Corollary 2.4.5 ([Yoneda Embedding]) *If* $\mathbf{C}$ *is locally small, the contravariant Yoneda embedding,* $\mathbf{y} : \mathbf{C}^{op} \to \mathbf{Set}^{\mathbf{C}}$ *defined by* $C \mapsto h^C$ *is a full embedding.*

Proof. We need to show that for the functor $\mathbf{y} : \mathbf{C}^{op} \to \mathbf{Set}^{\mathbf{C}}$ the induced mapping ψ is a bijection where:

$$\psi : Hom_{\mathbf{C}}(C, C') \to Hom_{\mathbf{Set}^{\mathbf{C}}}(\mathbf{y}(C'), \mathbf{y}(C))$$

defined such that:

$$f : C \to C' \mapsto \mathbf{y}f : \mathbf{y}(C') \dot{\to} \mathbf{y}(C)$$

where the natural transformation $\mathbf{y}f$ is defined, for $B \in |\mathbf{C}|$, by:

$$\begin{aligned} (\mathbf{y}f)_B : \ & \mathbf{y}(C')(B) \ \to \ \mathbf{y}(C)(B) \\ (\mathbf{y}f)_B : \ & h^{C'}(B) \ \to \ h^C(B) \end{aligned}$$

So $(\mathbf{y}f)_B = Hom_{\mathbf{C}}(f, B) = h_B(f)$, hence $\mathbf{y}f = Hom_{\mathbf{C}}(f, 1_{\mathbf{C}})$. However we know by the Yoneda Lemma, that

$$y : \mathbf{y}(C)(C') \to Hom_{\mathbf{Set}^{\mathbf{C}}}(\mathbf{y}(C'), \mathbf{y}(C))$$

is a bijection, (replace the functor F with the functor $\mathbf{y}(C)$ in the statement of the Yoneda Lemma and apply it to the object $C' \in |\mathbf{C}|$),defined such that:

$$f : C \to C' \mapsto \bar{f} : \mathbf{y}(C') \dot{\to} \mathbf{y}(C)$$

where the natural transformation $\bar{f}$ is defined such that $g : C' \to B \mapsto [\mathbf{y}(C)g](f)$. Now we show that $\psi = y$: Let $f : C \to C'$ and $g : C' \to B$

$$\begin{aligned} (yf)_B(g) = (\bar{f})_B(g) = h^C(g)(f) = (g)(f) = h_B(f)(g) \\ = (\mathbf{y}f)_B(g) = (\psi f)_B(g) \end{aligned}$$

Therefore $(yf)_B = (\psi f)_B$ for all $B \in |\mathbf{C}|$, hence $y = \psi$. We then have that the induced mapping ψ is the Yoneda Lemma bijection and therefore is a bijection, making $\mathbf{y}$ a full and faithful functor. □

2.4.3 Presheaf Categories

We can form a category of all presheaves over any category **C**:

Definition 2.4.6 Let **C** be a fixed small category. The functor category from **C** to **Set** is a category with:

(i) objects: functors $P : \mathbf{C} \to \mathbf{Set}$; and

(ii) arrows: natural transformations $\theta : P \to P'$ between such functors.

We call this the *category of presheaves*, denoted $\mathbf{Set}^{\mathbf{C}}$.

Example 2.4.7

(a) Let G be a group considered as a one-object category (see Example 2.4.2). We have previously seen that functors $P_X : \mathbf{C}_G \to \mathbf{Set}$ are representations of G. Therefore the objects of the category of presheaves are exactly those of the category of G-**Set**. As for arrows, an arrow in the category of presheaves is a natural transformation $\phi : P_X \to P_Y$. We now show that these will correspond exactly to the arrows of G-**Set**. The fact that ϕ is a natural transformation implies that we get the following commuting diagram:

$$\begin{array}{ccc} * & & P_X(*) \xrightarrow{\phi_*} P_Y(*) \\ \Big\downarrow g & & \Big\downarrow P_X g \qquad\qquad \Big\downarrow P_Y g \\ * & & P_X(*) \xrightarrow[\phi_*]{} P_Y(*) \end{array}$$

$$\text{i.e } P_Y g \circ \phi_* = \phi_* \circ P_X g$$

Hence,

$$P_Y g(\phi_*(x)) = \phi_*(P_X g(x))$$

$$g \cdot \phi_*(x) = \phi_*(g \cdot x)$$

Letting $X = P_X(*), Y = P_Y(*)$, and $g \cdot_1 x = P_X(g)(x)$, $g \cdot_2 y = P_Y(g)(y)$ we see $\phi = (\phi_*) : (X, \cdot_1) \to (Y, \cdot_2)$ is exactly the same as a G-set morphism given in Example 2.1.17. This implies that ϕ is an arrow for $\mathbf{Set}^{\mathbf{C}_G}$ if and only if ϕ is an arrow for G-**Set**. We can therefore conclude that G-**Set** is a presheaf category.

(b) **Set** is also a presheaf category. Let $\mathbf{C} = \mathbf{1}$, the category with one element and the identity arrow. Then a presheaf $P_X : \mathbf{1} \to \mathbf{Set}$ is a functor which on the object $*$, $P_X(*)$ gives rise to an arbitrary set. Therefore the objects of this presheaf category can be seen as elements of **Set**. Moreover the arrows of the presheaf category are natural transformations $\phi : P_X \to P_Y$, i.e., they are functions $\phi_* : X \to Y$, i.e. arrows in **Set** (the natural transformation condition is trivial in this case).

2.4.4 Properties of Functor Categories

We begin by stating results from [14].

Proposition 2.4.8 *If* $\mathbf{C}$ *is a small-complete category, so is every functor category* $\mathbf{C}^{\mathbf{D}}$, *for* $\mathbf{D}$, *any category.*

Corollary 2.4.9 *Any presheaf category is small-complete, where limits are given pointwise.*

In particular, in such categories, monomorphisms are given pointwise. We identify a monomorphism in **Set** with the associated inclusion.

Theorem 2.4.10 *For any small category* $\mathbf{C}$, *the functor category* $\mathbf{Set}^{\mathbf{C}}$ *is cartesian closed.*

We examine the proof given in [13].

Proof. We can observe that if the exponential exists for $F, G : \mathbf{C} \to \mathbf{Set}$, we ought to have

$$\begin{aligned}(F \Rightarrow G)(A) &\cong Nat(h^A, F \Rightarrow G), \text{ by Yoneda's Lemma 2.4.2} \\ &\cong Nat(h^A \times F, G), \text{ by adjointness}\end{aligned}$$

So we are led to define $(F \Rightarrow G)(A) = Nat(h^A \times F, G)$. Moreover, for $f : A \to B$ and $C \in |\mathbf{C}|$, $(F \Rightarrow G)(f) : (F \Rightarrow G)(A) \to (F \Rightarrow G)(B)$ is defined by

$$(((F \Rightarrow G)(f))(\theta))_C(g, c) = \theta_C(g \circ f, c),$$

for any $\theta \in (F \Rightarrow G)(A)$, $g : B \to C$ and $c \in FC$.
The counit functor $eval_{G,F} : (F \Rightarrow G) \times F \to G$ is defined by

$$eval_{G,F}(C)(\phi, c) = \phi_C(1_C, c)$$

for any $\phi \in (F \Rightarrow G)(C)$ and $c \in FC$.
The exponential adjunction

$$\frac{H \times F \xrightarrow{t} G}{H \xrightarrow{\lambda_t} (F \Rightarrow G)}$$

is defined by

$$((\lambda_t)_A)(a))_C(h, c) = t_C(H(h)(a), c),$$

for any objects $A, C \in |\mathbf{C}|, h : A \to C, a \in H(A)$ and $c \in FC$. □

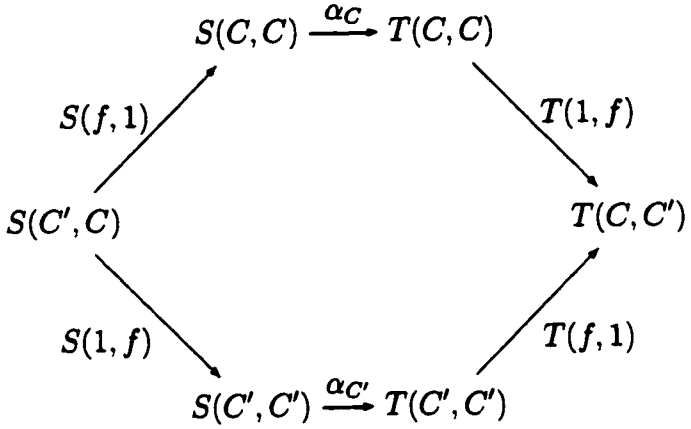
2.4.5 Day Tensor

To construct our tensor in our presheaf category we shall make use of a special construction due to Day [8]. However, before being able to explain this construct we must understand about dinatural transformations, ends and coends.

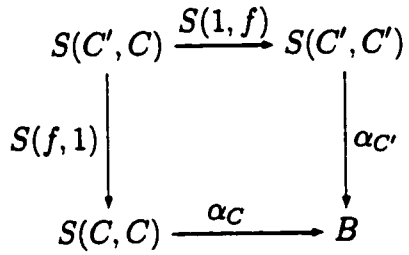
We begin with the notion of dinatural transformations, which link special kinds of bifunctors with mixed-variant domains, i.e. functors with domain $\mathbf{C}^{op} \times \mathbf{C}$.

Definition 2.4.11 Given categories $\mathbf{C}, \mathbf{B}$, and functors $S, T : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{B}$, a *dinatural transformation* $\alpha : S \xrightarrow{\bullet\bullet} T$ is a function which assigns to each object $C \in |\mathbf{C}|$ an arrow $\alpha_C : S(C, C) \to T(C, C)$ of $\mathbf{B}$, called the *component* of α at C, in such a way that for every arrow $f : C \to C'$ of $\mathbf{C}$ the following hexagonal diagram

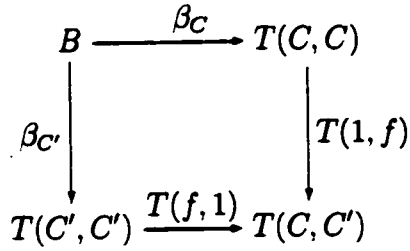
commutes.



Definition 2.4.12 If $T = B : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{B}$ is constant in both variables, a dinatural transformation $\alpha : S \xrightarrow{\bullet\bullet} B$ consists of components $\alpha_C : S(C,C) \to B$ which makes the following diagram commute, for every $f : C \to C'$:



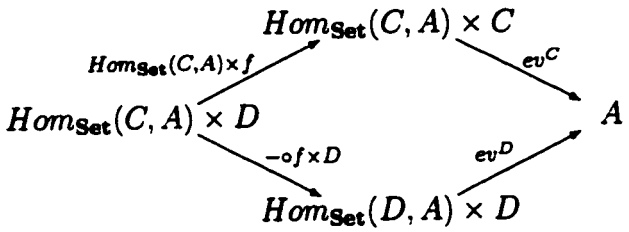
Such a transformation $\alpha : S \xrightarrow{\bullet\bullet} B$ is called a *co-wedge*[1] from S to B. A *wedge* $\beta : B \xrightarrow{\bullet\bullet} T$ is the dual concept to co-wedge, where $S = B : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{B}$ is constant in both variables, given by components $\beta_C : B \to T(C,C)$ such that for every $f : C \to C'$, the following is commutative.



We may examine an example of a co-wedge, taken from [14].

[1]Note: MacLane [14], ambiguously calls both wedges and co-wedges "wedges". We use categorical terminology, so ends are universal wedges and coends are universal co-wedges.

Example 2.4.13 A co-wedge, is given by the following family of arrows $\langle ev^B : Hom_{\mathbf{Set}}(B, A) \times B \to A\rangle_{B \in |\mathbf{Set}|}$ for $A \in |\mathbf{Set}|$ where the component ev^B is defined by $(g, b) \mapsto g(b)$. Given an arrow $f : D \to C$ in **Set** we construct the diagram



By chasing $(k, x) \in Hom_{\mathbf{Set}}(C, A) \times D$ in the diagram above

$$(k, x) \xmapsto{ev^C \circ Hom_{\mathbf{Set}}(C,A)\times f} k(f(x)), \text{and}$$

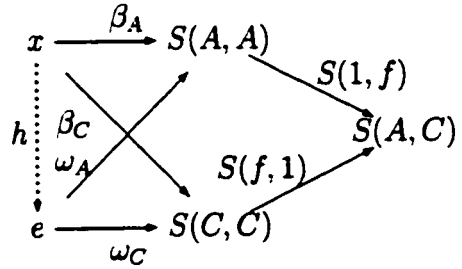
$$(k, x) \xmapsto{ev^D \circ (-\circ f \times D)} (k \circ f)(x) = k(f(x)).$$

Therefore $\langle ev^B \rangle_{B \in |\mathbf{Set}|}$ is a dinatural transformation from $(Hom_{\mathbf{Set}}(-_1, A) \times -_2)$ to the constant functor which always maps to the object A. From now on we denote a constant functor by its value.

From the notion of a wedge above we can define the *end* of a functor.

Definition 2.4.14 An *end* of a functor $S : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{X}$ is a universal dinatural transformation from a constant functor e to S; that is, an end of S is a pair $\langle e, \omega \rangle$, where e is an object of $\mathbf{X}$ and $\omega : e \xrightarrow{\bullet\bullet} S$ is a wedge with the property that to every wedge $\beta : x \xrightarrow{\bullet\bullet} S$ there is a unique arrow $h : x \to e$ with $\beta_A = \omega_A h$ for all $A \in |\mathbf{C}|$.

Thus for each arrow $f : A \to C$ of $\mathbf{C}$ there is a diagram



such that both quadrilaterals commute: the universal property of ω states that there is a unique h such that both triangles (at the left) commute.

Definition 2.4.15 The uniqueness property, states that if $\langle e, \omega \rangle$ and $\langle e', \omega' \rangle$ are two ends of S, there is a unique isomorphism $u : e \to e'$ with $\omega' \circ u = \omega$. We call ω the *universal wedge*, with *components* ω_c, and is written with integral notation as

$$e = \int_C S(C,C) = \text{ End of } S$$

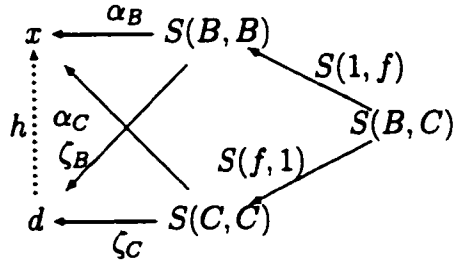
We construct the end for set-valued functors as follows:

$$\int_C S(C,C) = \{g \in \prod_C S(C,C); \text{ for any wedge } \beta : a \xrightarrow{\cdot\cdot} S,\ S(1,f) \circ \beta_B = S(f,1) \circ \beta_C, \text{ for all } B, C \in |\mathbf{C}| \text{ and } f : B \to C \text{ in } \mathbf{C}\}$$

Notice that an end is a limit - a subobject of a product. Moreover, an analogous construction of ends can be given for arbitrary complete categories.

However, the Day construction is based on the dual version of an end, called a *coend.*

Definition 2.4.16 A *coend* of a functor $S : \mathbf{C}^{op} \times \mathbf{C} \to \mathbf{X}$ is a universal dinatural transformation from S to a constant d; that is, a coend of S is a pair $\langle d, \zeta : S \xrightarrow{\cdot\cdot} d \rangle$, where d is an object of $\mathbf{X}$ and $\zeta : S \xrightarrow{\cdot\cdot} d$ is a co-wedge with the property that to every co-wedge $\alpha : S \xrightarrow{\cdot\cdot} x$ there is a unique arrow $h : d \to x$ with $\alpha_A = h \circ \zeta_A$ for all $A \in |\mathbf{C}|$.
Thus for each arrow $f : C \to B$ of $\mathbf{C}$ there is a diagram



such that both quadrilaterals commute. The universal property of ζ states that there is a unique h such that both triangles commute.

We then construct set-valued coends as follows:

$$\int^C S(C,C) = \{g \in \sum_C S(C,C); \text{ for any co-wedge } \alpha : S \xrightarrow{\cdot\cdot} x,\ \alpha_B \circ S(1,f) = \alpha_C \circ S(f,1), \text{ for all } B, C \in |\mathbf{C}| \text{ and } f : C \to B \text{ in } \mathbf{C}\}/\sim$$

Where the equivalence relation $\sim$ is such that:

$$g \sim g' \text{ iff for each arrow } f : C \to B,$$

$$\zeta_B \circ S(1, f)(g) = \zeta_B \circ S(1, f)(g') \text{ and } \zeta_C \circ S(f, 1)(g) = \zeta_C \circ S(f, 1)(g')$$

Again, this construction works for general cocomplete categories.

Definition 2.4.17 Given the category **Set**, with the usual cartesian closed structure $(\mathbf{Set}, \times, 1, \Rightarrow)$, and a monoidal category $(\mathbf{C}, \otimes_C, I_C)$ (not necessarily closed), we can define a monoidal closed structure $(\mathbf{Set}^{\mathbf{C}}, \otimes_{Day}, I_{Day}, \multimap_{Day})$ on the functor category $\mathbf{Set}^{\mathbf{C}}$. The construction, due to Brian Day [8], called the *Day tensor*, is defined as follows: Let $S, T \in |\mathbf{Set}^{\mathbf{C}}|$, and $s, X, Y \in |\mathbf{C}|$

$$\begin{aligned}(S \otimes_{Day} T)(s) &= \int^X SX \times \int^Y TY \times Hom_{\mathbf{C}}(X \otimes_{\mathbf{C}} Y, s) \\ &= \int^{X,Y} SX \times TY \times Hom_{\mathbf{C}}(X \otimes_{\mathbf{C}} Y, s) \\ &= \sum_{X,Y} SX \times TY \times Hom_{\mathbf{C}}(X \otimes_{\mathbf{C}} Y, s), \\ &\quad \text{under an equivalence relation}\end{aligned}$$

The equivalence relation is generated from the following relation:

$$(x \in SX, y \in TY, \eta : X \otimes_{\mathbf{C}} Y \to s) \sim (x' \in SX', y' \in TY', \eta' : X' \otimes_{\mathbf{C}} Y' \to s)$$

if and only if $\exists f : X' \to X$ and $g : Y' \to Y$, such that $Sf(x') = x$, $Tg(y') = y$ and $\eta \circ (f \otimes_{\mathbf{C}} g) = \eta'$.

The linear function space is given by $(A \multimap_{Day} C)(s) = Hom_{\mathbf{Set}^{\mathbf{C}}}(A, C(s \otimes_{\mathbf{C}} -))$. On a map $f : s \to s'$, we define $(A \multimap_{Day} C)(f)(\theta)$ such that

$$((A \multimap_{Day} C)(f)(\theta))_{s''}(a) = C(f \otimes_C id_{s''})(\theta_{s''}(a))$$

Proposition 2.4.18 *The functor* $(A \multimap_{Day} -)$ *is left adjoint to the Day tensor* $(- \otimes_{Day} A)$.

Proof. We construct an isomorphism of the form

$$Hom(B \otimes_{Day} A, C) \cong Hom(B, A \multimap_{Day} C),$$

by defining two maps, $\alpha : Hom(B \otimes_{Day} A, C) \rightarrow Hom(B, A \multimap_{Day} C)$ and $\beta : Hom(B, A \multimap_{Day} C) \rightarrow Hom(B \otimes_{Day} A, C)$ which are inverses to each other.

Define the maps as follows:

$$\begin{array}{rccc} \alpha: & Hom(B \otimes_{Day} A, C) & \rightarrow & Hom(B, A \multimap_{Day} C) \\ & \theta & \mapsto & \theta^{\alpha} \end{array}$$

such that

$$(\theta^{\alpha}_{s}(b))_{s'}(a) = \theta_{s+s'}(b, a, id_{s+s'})$$

where $b \in B(s)$ and $a \in A(s')$.

$$\begin{array}{rccc} \beta: & Hom(B, A \multimap_{Day} C) & \rightarrow & Hom(B \otimes_{Day} A, C) \\ & \psi & \mapsto & \psi^{\beta} \end{array}$$

such that

$$\psi^{\beta}_{s}(b_X, a_X, \eta : X + Y \rightarrow s) = C\eta((\psi_X(b_X))_Y(a_Y))$$

where $b_X \in B(X)$ and $a_Y \in A(Y)$.

First we show that β is well-defined. Suppose $(b_X, a_Y, \eta : X + Y \rightarrow s) \sim (b_{X'}, a_{Y'}, \eta' : X' + Y' \rightarrow s)$, then by definition there exists maps $f : X' \rightarrow X$ and $g : Y' \rightarrow Y$, such that $Bf(b_{X'}) = b_X$, $Ag(a_{Y'}) = a_Y$ and $\eta' \circ [f, g] = \eta$. Therefore, we get the following equality by using naturality of $\psi_X(b_X)$ and ψ.

$$\begin{aligned} C\eta((\psi_X(b_X))_Y(a_Y)) &= C\eta(C[id_x, g]((\psi_X(b_X))_{Y'}(a_{Y'}))) \\ &= C\eta(C[id_X, g]((A \multimap C)(f)((\psi_{X'}(b_{X'}))_{Y'}(a_{Y'})))) \\ &= C\eta(C[id_X, g](C[f, id_{Y'}]((\psi_{X'}(b_{X'}))_{Y'}(a_{Y'})))) \\ &= C\eta(C[f, g]((\psi_{X'}(b_{X'}))_{Y'}(a_{Y'}))) \\ &= C(\eta \circ [f, g])((\psi_{X'}(b_{X'}))_{Y'}(a_{Y'})) \\ &= C\eta'((\psi_{X'}(b_{X'}))_{Y'}(a_{Y'})) \end{aligned}$$

Secondly we show that the maps are inverses of each other. We begin by showing that $(\alpha \circ \beta)(\psi) = \psi$.

$$(((\alpha \circ \beta)(\psi))_s(b))_{s'}(a) = \psi^{\beta}_{s+s'}(b, a, id_{s+s'})$$

$$\begin{aligned} &= Cid_{s+s'}((\psi_s(b))_{s'}(a)) \\ &= id_{C(s+s')}((\psi_s(b))_{s'}(a)) \\ &= ((\psi_s(b))_{s'}(a)) \end{aligned}$$

Now we show that $(\beta \circ \alpha)(\theta) = \theta$.

$$\begin{aligned} ((\beta \circ \alpha)(\theta))_s(b_X, a_Y, \eta : X + Y \to s) &= C\eta((\theta^{\alpha}_X(b_X))_Y(a_Y)) \\ &= C\eta(\theta_{X+Y}(b, a, id_{X+Y})) \\ &= \theta_s \circ (B \otimes_{Day} A)(\eta)(b, a, id_{X+Y}) \\ &= \theta_s(b, a, \eta : X + Y \to s) \end{aligned}$$

□

The identity element I_{Day} is defined to be $I_{Day} = Hom_{\mathbf{C}}(I_{\mathbf{C}}, -)$.

We justify that I_{Day} is in fact the identity element for our Day tensor. Let $S \in |\mathbf{Set}^{\mathbf{C}}|$. We show that $S \otimes_{Day} I_{Day} \cong S$:

$$\begin{aligned} (S \otimes_{Day} I_{Day})(s) &= \textstyle\sum_{X,Y} SX \times I_{Day}(Y) \times Hom_{\mathbf{C}}(X \otimes_{\mathbf{C}} Y, s) \\ &= \textstyle\sum_{X,Y} SX \times Hom_{\mathbf{C}}(I_{\mathbf{C}}, Y) \times Hom_{\mathbf{C}}(X \otimes_{\mathbf{C}} Y, s) \\ &\quad \text{under an equivalence relation} \end{aligned}$$

However, we will show that an arbitrary element $(x, t, \eta) \in \sum_{X,Y} SX \times Hom_{\mathbf{C}}(I_{\mathbf{C}}, Y) \times Hom_{\mathbf{C}}(X \otimes_{\mathbf{C}} Y, s)$ is equivalent to the element $(S(\eta \circ (id_X \otimes_{\mathbf{C}} t))(x), id_{I_{\mathbf{C}}}, id_s)$.

(i) $(x, t, \eta) \sim (x, id_{I_{\mathbf{C}}}, \eta \circ (id_X \otimes_{\mathbf{C}} t))$. Pick the maps $id_X : X \to X$ and $t : I_{\mathbf{C}} \to Y$. Observe that they satisfy the requirements for the equivalence,

$$S(id_X)(x) = id_{S(X)}(x) = x$$

$$I_{Day}(t)(id_{I_{\mathbf{C}}}) = t \circ id_{I_{\mathbf{C}}} = t$$

$$\eta \circ (id_X \otimes_{\mathbf{C}} t) = \eta \circ (id_X \otimes_{\mathbf{C}} t)$$

(ii) $(S(\eta \circ (id_X \otimes_{\mathbf{C}} t))(x), id_{I_{\mathbf{C}}}, id_s) \sim (x, id_{I_{\mathbf{C}}}, \eta \circ (id_X \otimes_{\mathbf{C}} t))$. Pick the maps $\eta \circ (id_X \otimes_{\mathbf{C}} t) : X \to s$ and $id_{I_{\mathbf{C}}} : I_{\mathbf{C}} \to I_{\mathbf{C}}$. Observe that they satisfy the requirements for the equivalence,

$$S(\eta \circ (id_X \otimes_{\mathbf{C}} t))(x) = S(\eta \circ (id_x \otimes_{\mathbf{C}} t))(x)$$

$$I_{Day}(id_{I_{\mathbf{C}}})(id_{I_{\mathbf{C}}}) = id_{I_{\mathbf{C}}} \circ id_{I_{\mathbf{C}}} = id_{I_{\mathbf{C}}}$$

$$id_s \circ (\eta \circ (id_X \otimes_{\mathbf{C}} t) \otimes_{\mathbf{C}} id_{I_{\mathbf{C}}}) = \eta \circ (id_X \otimes_{\mathbf{C}} t)$$

By transitivity we get $(x, t, \eta) \sim (S(\eta \circ (id_X \otimes_{\mathbf{C}} t))(x), id_{I_{\mathbf{C}}}, id_s)$, which allows us to define the following maps:

$$\begin{array}{rccc} f: & (S \otimes_{Day} I_{Day})(s) & \to & S(s) \\ & (x,t,\eta) \sim (S(\eta \circ (id_X \otimes_{\mathbf{C}} t))(x), id_{I_{\mathbf{C}}}, id_s) & \mapsto & S(\eta \circ (id_X \otimes_{\mathbf{C}} t))(x) \end{array}$$

and the map

$$\begin{array}{rccc} g: & S(s) & \to & (S \otimes_{Day} I_{Day})(s) \\ & s & \mapsto & (s, id_{I_{\mathbf{C}}}, id_s) \end{array}$$

One can easily verify that $f \circ g = id_{(S \otimes_{Day} I_{Day})(s)}$ and $g \circ f = id_{S(s)}$, confirming the existence of the isomorphism.

If $\mathbf{C}$ is a smcc, this suggests considering the presheaf category $\mathbf{Set}^{\mathbf{C}}$ with both its standard cartesian structure, as well as the induced Day smcc structure. Therefore, we have the following proposition from [19].

Proposition 2.4.19 *If* $\mathbf{C}$ *is a monoidal category, then* $\mathbf{Set}^{\mathbf{C}}$ *is a bicartesian dcc.*

Corollary 2.4.20 *If* $\mathbf{C}$ *is a symmetric monoidal category, then* $\otimes_{Day}$*, the tensor product of* $\mathbf{Set}^{\mathbf{C}}$ *is also symmetric.*

2.5 Monads

As stated in [4], from one point of view, a monad is an abstraction of certain properties of algebraic structures. From another point of view, it is an abstraction of certain

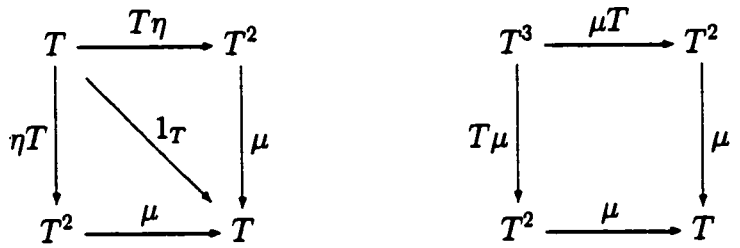
properties of adjoint functors. We shall use monads in order to add non-determinism into our model for the finite π-calculus. Therefore, we construct a specific monad called the free-semilattice monad, which on a presheaf category is representable by the finite powerset monad.

Definition 2.5.1 A *monad* (T, η, μ) on a category $\mathbf{C}$ consists of a functor $T : \mathbf{C} \to \mathbf{C}$ and natural transformations $\eta : 1_A \to T$ and $\mu : T^2 \to T$ satisfying the equations:

(i) $\mu \circ T\eta = 1_T = \mu \circ \eta T$

(ii) $\mu \circ \mu T = \mu \circ T\mu$

These equations are sometimes called the *unity laws* and *associative laws* respectively and are illustrated by the following commutative diagrams:



Example 2.5.2 Let $T = \mathcal{P}_f$ be the covariant finite powerset functor **Set** $\to$ **Set**, that is for any set A,

$$\mathcal{P}_f(A) = \{X | X \subseteq A \text{ and } X \text{ is a finite set}\}$$

and, for any mapping $g : A \to B$, and any finite subset $X \subseteq A$,

$$\begin{aligned} \mathcal{P}_f g(X) &= g[X] \\ &= \{g(x) | x \in X\} \end{aligned}$$

Furthermore, let the natural transformations η and μ be given by the mappings $\eta(A) : A \to \mathcal{P}_f(A)$ and $\mu(A) : \mathcal{P}_f(\mathcal{P}_f(A)) \to \mathcal{P}_f(A)$ defined by:

1. $\eta(A)(a) = \{a\}$
2. $\mu(A)(\mathcal{X}) = \bigcup \mathcal{X} = \bigcup_{X \in \mathcal{X}} X$

for any set A, any element $a \in A$ and any finite set $\mathcal{X}$ of finite subsets of A. Finally, we show that the natural transformations satisfy the appropriate equations:

(i)

$$\begin{aligned}
\mu_X \circ \mathcal{P}_f(\eta_X)(\mathcal{X}) &= \mu_X(\{\eta_X(x); x \in \mathcal{X}\}) \\
&= \mu_X(\{\{x\}; x \in \mathcal{X}\}) \\
&= \cup_{x \in \mathcal{X}}\{x\} \\
&= \mathcal{X}
\end{aligned}$$

where $\mathcal{X} \in (\mathcal{P}_f)^2(X)$. Similarly, one can check that $\mu_X \circ \eta_{(\mathcal{P}_f(X))} = id_{\mathcal{P}_f}$

(ii)

$$\begin{aligned}
\mu_X \circ \mu_{(\mathcal{P}_f(X))}(\mathcal{A}) &= \mu_X(\cup_{\mathcal{X} \in \mathcal{A}}\mathcal{X}) \\
&= \cup_{X \in (\cup_{\mathcal{X} \in \mathcal{A}}\mathcal{X})}X \\
&= \cup_{\mathcal{X} \in \mathcal{A}}\cup_{X \in \mathcal{X}}X \\
&= \mu_X(\{\cup_{X \in \mathcal{X}}X; \mathcal{X} \in \mathcal{A}\}) \\
&= \mu_X(\{\mu_X(\mathcal{X}); \mathcal{X} \in \mathcal{A}\}) \\
&= \mu_X \circ \mathcal{P}_f(\mu_X)(\mathcal{A})
\end{aligned}$$

where $\mathcal{A} \in (\mathcal{P}_f)^3(X)$.

Thus, we indeed have a valid monad on **Set**.

2.6 Free-Semilattice Monad for Set

To have non-deterministic choice within our model of the finite π-calculus, we shall need to create a monad to approximate that behavior within our presheaf category.

First, we shall need to define the category of complete $\vee$-semilattices and complete $\vee$-semilattice homomorphisms, called **CSLat**.

Definition 2.6.1

1. A $\vee$-*semilattice* is an ordered set A, together with the maps $\vee : A \times A \to A$ and $\perp : 1 \to A$ satisfying the following equations:

   (i) $a \vee a = a$

   (ii) $a \vee b = b \vee a$

   (iii) $a \vee (b \vee c) = (a \vee b) \vee c$

   (iv) $a \vee \perp = a$

   The order is defined such that $A \leq b$ if and only if $A \vee b$.

2. A $\vee$-semilattice A is *complete* if every subset $S \subseteq A$ has a suprema, i.e. $\vee S$ exists for any $S \subseteq A$.

3. A *complete* $\vee$-*semilattice homomorphism* $\phi : A \to B$ is a map which preserves suprema, i.e. $\phi(\vee S) = \vee(\phi S)$, where $S \subseteq A$. This condition preserves the ordering on A.

Our candidate for a monad to which represents nondeterminism in our model is the free-semilattice monad, which is induced by the forgetful functor $U : \mathbf{CSLat} \to \mathbf{Set}$ from the category of complete $\vee$-semilattices and complete $\vee$-semilattice homomorphisms to the category **Set** and its corresponding left adjoint $F : \mathbf{Set} \to \mathbf{CSLat}$.

The construction of this monad will be given using the Eilenberg-Moore category construction. We will show through this construction that the finite powerset monad $\mathcal{P}_f$ given in Example 2.5.2, is the free-semilattice monad for the category **Set**.

The following are an adaptation from [13].

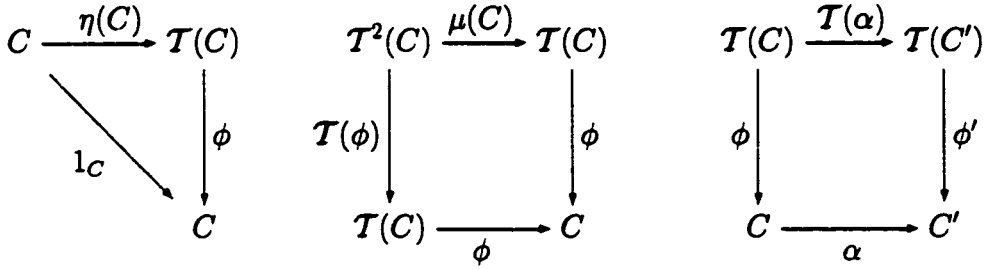
Definition 2.6.2 Given a monad $(\mathcal{T}, \eta, \mu)$ on a category $\mathbf{C}$, the *Eilenberg-Moore category* $\mathbf{C}^{\mathcal{T}}$ of the monad is defined as follows. Its objects called *algebras*, are pairs (C, ϕ), where $\phi : \mathcal{T}(C) \to C$ is an arrow of $\mathbf{C}$ satisfying the equations

$$\phi\eta(C) = 1_C, \quad \phi\mu(X) = \phi\mathcal{T}(\phi)$$

for all objects $C \in |\mathbf{C}|$. Its arrows, called *homomorphisms*, $(C, \phi) \to (C', \phi')$ are arrows $\alpha : C \to C'$ of $\mathbf{C}$ satisfying the equation

$$\phi'\mathcal{T}(\alpha) = \alpha\phi$$

These equations are illustrated by the following commutative diagrams:



Proposition 2.6.3 *The Eilenberg-Moore category* $\mathbf{Set}^{\mathcal{P}_f}$ *is the category* **CSLat**.

Proof. An object of $\mathbf{Set}^{\mathcal{P}_f}$ is a pair (X, ϕ), where $X \in |\mathbf{Set}|$ and $\phi : \mathcal{P}_f(X) \to X$, which satisfies the equations

$$\phi(\{x\}) = x, \forall x \in X \quad \phi(\cup_{X \in \mathcal{X}} X) = \phi(\{\phi(X); X \in \mathcal{X}\})$$

with $\mathcal{X} \in \mathcal{P}_f(\mathcal{P}_f(X))$. In truth, we will show that an object of the Eilenberg-Moore category $\mathbf{Set}^{\mathcal{P}_f}$ represents a complete $\vee$-semilattice.

We begin by showing that the relation on X such that for $x, y \in X, x \leq y$ if and only if $\phi(\{x, y\}) = y$ is a genuine partial order. We need to show the following:

(i) reflexive: $x \leq x$, since $h(\{x\}) = x$ by the algebra equalities.

(ii) transitive: Suppose $x \leq y$ and $y \leq z$, this implies that $\phi(\{x, y\}) = y$ and $\phi(\{y, z\}) = z$. Hence by the algebra equalities we have that $z = \phi(\{y, z\}) = \phi(\{\phi(\{x, y\}), \phi(\{y, z\})\}) = \phi(\{x, y, z\}) = \phi(\{\phi(\{x\}), \phi(\{y, z\}) = \phi(\{x, z\})$. Thus, we get that $x \leq z$.

(iii) Suppose that $x \leq y$ and $y \leq x$, hence we get that $\phi(\{x, y\}) = y$ and $\phi(\{y, x\}) = x$. Since ϕ is well-defined, we have that $x = y$.

Next, we show that for any finite subset $S \subseteq X$, there exists a supremum $\vee(S) = \phi(S)$.

(a) First, we show that $\phi(S)$ is an upper bound for the subset S. Pick an arbitrary $x \in S$, we calculate $\phi(\{x, \phi(S)\})$:

$$\phi(\{x, \phi(S)\}) = \phi(\{\phi(\{x\}), \phi(S)\}) = \phi(\{x\} \cup S) = \phi(S)$$

Since, x was an arbitrary element of S, we have proven that for any $x \in S$, $\phi(\{x, \phi(S)\}) = \phi(S)$, which implies $x \leq \phi(S)$.

(b) Suppose there exists another upper bound for the subset S, say u. We will show that $\phi(S) \leq u$. Let $S = \{x_1, x_2, \ldots, x_n\}$

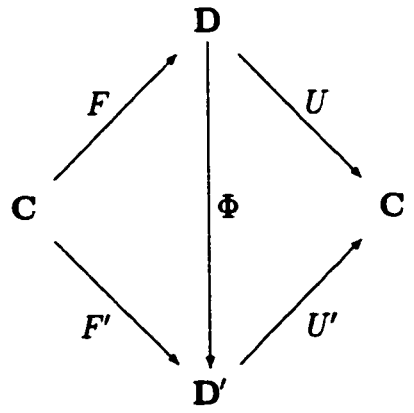
$$\begin{aligned}
\phi(\{\phi(S), u\}) &= \phi(\{\phi(S), \phi(\{u\})\}) \\
&= \phi(S \cup \{u\}) \\
&= \phi(\{x_1, \phi(\{\phi(\{x_2, \ldots \phi(\{\phi(\{x_n\}), \phi(\{u\})\}) \ldots\})\})\}) \\
&= u
\end{aligned}$$

Therefore $\phi(S) = \vee S$. What is left to show is that an arrow between the algebras is in fact a $\vee$-semilattice homomorphism. Hence, suppose $\phi : (X, \phi) \to (Y, \psi)$, hence an arrow $\alpha : X \to Y$ such that $\psi \mathcal{P}_f(\alpha) = \alpha\phi$. Therefore for any subset $S \subseteq X$, we get

$$\begin{aligned}
\psi \mathcal{P}_f(\alpha)(S) &= \alpha\phi(S) \\
\psi(\{\alpha(x); x \in S\}) &= \alpha(\phi(S)) \\
\psi(\alpha(S)) &= \alpha(\phi(S)) \\
\vee(\alpha(S)) &= \alpha(\vee S)
\end{aligned}$$

Therefore, the arrow preserves the suprema of arbitrary subsets, hence it is a $\vee$-semilattice homomorphism. □

Definition 2.6.4 Given a monad $(\mathcal{T}, \eta, \mu)$ on a category $\mathbf{C}$, by a *resolution* of this monad we mean a category $\mathbf{D}$ and a pair of adjoint functors $\mathbf{C} \xrightarrow{F} \mathbf{D} \xrightarrow{U} \mathbf{C}$ such that $UF = \mathcal{T}$ with adjunctions η (as given) and *eval* such that $UevalF = \mu$. The resolutions of the given monad form a category whose arrows
$\Phi : (\mathbf{D}, U, F, eval) \to (\mathbf{D}', U', F', eval')$ are functors $\Phi : \mathbf{D} \to \mathbf{D}'$ such that $\Phi F = F', U'\Phi = U$ and $\Phi eval = eval'\Phi$. In particular, the following two triangles commute:



Proposition 2.6.5 *The Eilenberg-Moore category* $\mathbf{C}^{\mathcal{T}}$ *of the monad* $(\mathcal{T}, \eta, \mu)$ *on* $\mathbf{C}$ *gives rise to a resolution* $(\mathbf{C}^{\mathcal{T}}, U^{\mathcal{T}}, F^{\mathcal{T}}, eval^{\mathcal{T}})$, *which is a terminal object in the category of all resolutions. Thus, given any resolution* $(\mathbf{D}, U, F, eval)$ *there is a unique functor* $K^{\mathcal{T}} : \mathbf{D} \to \mathbf{C}^{\mathcal{T}}$, *called the* comparison functor, *such that* $K^{\mathcal{T}}F = F^{\mathcal{T}}, U^{\mathcal{T}}K^{\mathcal{T}} = U$ *and* $K^{\mathcal{T}}eval = eval^{\mathcal{T}}K^{\mathcal{T}}$. *Moreover,* $U^{\mathcal{T}}$ *is faithful.*

The proof of this proposition, as seen in [13], is interesting because it gives a concrete construction of the functors $F^{\mathcal{T}}$, $U^{\mathcal{T}}$ and the natural transformation $eval^{\mathcal{T}}$. Therefore, we shall reproduce the proof in part here, to see their construction.

Proof.

(i) We define $U^{\mathcal{T}} : \mathbf{C}^{\mathcal{T}} \to \mathbf{C}$ by

$$U^{\mathcal{T}}(C, \phi) = C, \quad U^{\mathcal{T}}(\alpha) = \alpha$$

for any algebra (C, ϕ) and any homomorphism α. Evidently, $U^{\mathcal{T}}$ is faithful.

(ii) We define $F^{\mathcal{T}} : \mathbf{C} \to \mathbf{C}^{\mathcal{T}}$ by

$$F^{\mathcal{T}}(C) = (\mathcal{T}(C), \mu(C)), \quad F^{\mathcal{T}}(f) = \mathcal{T}(f)$$

for any object C and any arrow f of $\mathbf{C}$. It is easily checked that $(\mathcal{T}(C), \mu(C))$ is an algebra, that $\mathcal{T}(f)$ is a homomorphism and that $U^{\mathcal{T}}F^{\mathcal{T}} = \mathcal{T}$.

(iii) We define the natural transformation $eval^{\mathcal{T}}$ from $F^{\mathcal{T}}U^{\mathcal{T}}$ to the identity functor on $\mathbf{C}^{\mathcal{T}}$ by its action on the algebra (C, ϕ) as follows: the homomorphism $eval^{\mathcal{T}}(C, \phi) = \phi$. Indeed, the square

$$\begin{array}{ccc} \mathcal{T}^2(C) & \xrightarrow{\mathcal{T}(\phi)} & \mathcal{T}(C) \\ {\scriptstyle \mu(C)}\downarrow & & \downarrow{\scriptstyle \phi} \\ \mathcal{T}(C) & \xrightarrow[\phi]{} & C \end{array}$$

commutes.

□

Example 2.6.6 If we consider once again our finite powerset monad $\mathcal{P}_f$, we can construct the following terminal resolution

$$(\mathbf{Set}^{\mathcal{P}_f}, F^{\mathcal{P}_f}, U^{\mathcal{P}_f}, eval^{\mathcal{P}_f}) = (\mathbf{CSLat}, F^{\mathcal{P}_f}, U, eval^{\mathcal{P}_f})$$

where $U : \mathbf{CSLat} \rightarrow \mathbf{Set}$, is the usual forgetful functor, and where
$F^{\mathcal{P}_f} : \mathbf{Set} \rightarrow \mathbf{CSLat}$, is the functor defined such that $F^{\mathcal{P}_f}(X) = (\mathcal{P}_f(X), \mu(X))$.

Therefore, the Proposition 2.6.5 shows how the Eilenberg-Moore category is used to construct a left adjoint to the forgetful functor $U : \mathbf{CSLat} \rightarrow \mathbf{Set}$, which induces the finite powerset monad.

$$\mathbf{Set} \overset{F^{\mathcal{P}_f}}{\rightarrow} \mathbf{CSLat} \overset{U}{\rightarrow} \mathbf{Set}$$

Definition 2.6.7 Hence we shall call $\mathcal{P}_f$ the *free-semilattice monad* on the category **Set**, since it represents the left adjoint to the forgetful functor from **CSLat** to **Set**.

There is a more abstract construction which generalizes our approach to constructing the free-semilattice monad for an arbitrary **Cpo**-enriched cartesian category, [2, 10]. However, for our model for the finite π-calculus, it is unnecessary to generalize so.

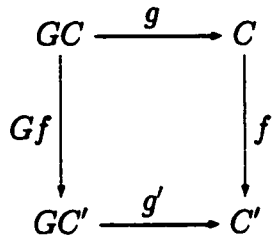
2.7 Fixed Points

The object of processes in our model will arise as a fixed point of a certain endofunctor in our model. Therefore, in this section we clarify what it means for a functor to have a fixed point by adapting the material which appears in [3].

2.7.1 Algebras for an Endofunctor

Definition 2.7.1 Let $\mathbf{C}$ be a category and $G : \mathbf{C} \rightarrow \mathbf{C}$ be an endofunctor. A *G-algebra* is a pair (C, g) where $g : GC \rightarrow C$ is an arrow of $\mathbf{C}$. A homomorphism

between G-algebras (C, g) and (C', g') is an arrow $f : C \to C'$ such that the following diagram commutes.



This construction gives rise to the category $(G : \mathbf{C})$ of G-algebras.

Definition 2.7.2 An object (C, g) of $(G : \mathbf{C})$ for which the arrow g is an isomorphism is a *fixed point* for G.

Based on the perception that a category is a generalized poset, where the ordering is given by arrows between objects, the following definition is reasonable.

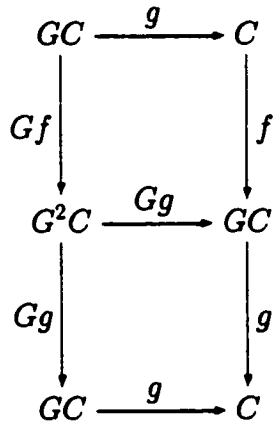
Definition 2.7.3 A *least fixed point* of a functor $G : \mathbf{C} \to \mathbf{C}$ is an initial object of $(G : \mathbf{C})$.

2.7.2 Lambek's Lemma

For Definition 2.7.3 to be non vacuous, we show that the initial object in the category of G-algebras (when it exists) is in fact a fixed point for the endofunctor G. This can be shown by using Lambek's Lemma below.

Theorem 2.7.4 (Lambek) *Let $G : \mathbf{C} \to \mathbf{C}$ be an endofunctor. If (C, g) is initial in the category $(G : \mathbf{C})$, then g is an isomorphism.*

Proof. Suppose (C, g) is initial, (GC, Gg) is an object of the category $(G : \mathbf{C})$ and hence there is a unique arrow $f : (C, g) \to (GC, Gg)$. This means that the top square of the diagram commutes.



The bottom square also commutes, since trivially $Gg \circ g = Gg \circ g$. Thus the whole rectangle commutes and so $g \circ f : (C, g) \to (C, g)$ is an arrow between G-algebras. But (C, g) is initial and so only has the identity endomorphism. Thus $g \circ f = id$. Then the commutativity of the upper square gives us that

$$f \circ g = Gg \circ Gf = G(g \circ f) = G(id) = id_G$$

which means that $f = g^{-1}$ and that g is an isomorphism. □

Next we examine the question of when such fixed points exist.

2.7.3 Fixed Points in $\mathbf{Set}^{\mathbf{Inj}}$

The following results describe sufficient conditions in $\mathbf{Set}^{\mathbf{C}}$ which allow the functor $G : \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{\mathbf{C}}$ to admit a least fixed point.

Lemma 2.7.5 *Let $A_0 \xrightarrow{\alpha_0} A_1 \xrightarrow{\alpha_1} A_2 \xrightarrow{\alpha_2} \cdots$ be a countable sequence of objects in $\mathbf{Set}^{\mathbf{C}}$, where $\alpha_i : A_i \xrightarrow{\cdot} A_{i+1}$ are monomorphisms. Then, the colimit of the given sequence exists and is represented by the pointwise directed union of the objects of the sequence, i.e. $((\cup_i)A_i)(C) = \cup_i(A_i(C))$ for each object $C \in |\mathbf{C}|$.*

Proof. We begin by proving that $\cup_i A_i$ is an element of $\mathbf{Set}^{\mathbf{C}}$. Recall the discussion after Corollary 2.4.9 that monomorphisms are given pointwise. Let $C \in |\mathbf{C}|$. We

note that if $x \in A_i(C)$ then due to the monomorphism (= inclusion) there is a corresponding x in every $A_j(C)$ where $j \geq i$. Hence this signifies that if $x \in (\cup_i A_i)(C)$ then there is a minimal i_0 such that $x \in A_{i_0}(C)$. This fact is important for our arguments below.

(i) On objects; $(\cup_i A_i)(C) = \cup_i(A_i(C))$

(ii) On morphisms $f : C \to C'$ we define

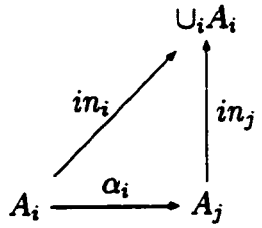
$$\begin{array}{rccl} (\cup_i A_i)(f): & (\cup_i A_i)(C) & \to & (\cup_i A_i)(C') \\ & x & \mapsto & A_{i_0}(f)(x), \text{ where } i_0 = min\{i; x \in A_i(C)\ \} \end{array}$$

(a) $(\cup_i A_i)(id_C)(x) = A_{i_0}(id_c)(x) = id_{A_{i_0}(C)}(x) = x = id_{(\cup_i A_i)(C)}(x)$

(b) Let $f : C \to C'$ and $g : C' \to C''$, $(\cup_i A_i)(g \circ f)(x) = A_{i_0}(g \circ f)(x) = A_{i_0}(g) \circ A_{i_0}(f)(x) = (\cup_i A_i)(g) \circ (\cup_i A_i)(f)(x)$, where i_0 is as above.

Since we have satisfied all the required properties, we have that $\cup_i A_i \in \mathbf{Set}^{\mathbf{C}}$.

We now show that $\cup_i A_i$ is the colimit of the given sequence. Hence, we must show that there exist maps $in_i : A_i \stackrel{\cdot}{\to} \cup_i A_i$ such that the following diagram commutes:

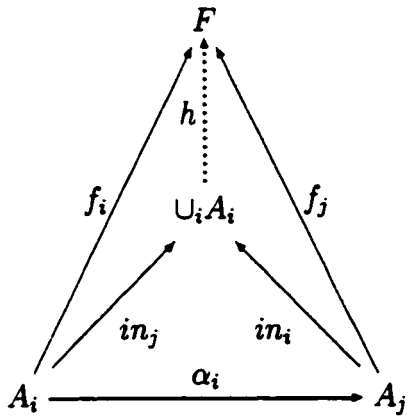


First, define the maps $in_j : A_j \stackrel{\cdot}{\to} \cup_i A_i$ such that for any $C \in |\mathbf{C}|$

$$\begin{array}{rccl} (in_j)_C: & A_j(C) & \to & (\cup_i A_i)(C) \\ & x & \mapsto & x \end{array}$$

Now, suppose a family of maps $f_j : A_j \stackrel{\cdot}{\to} F$, we must show there exists a unique map

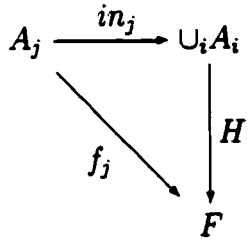
$h : \cup_i A_i \stackrel{\cdot}{\to} F$. Which makes the following diagram commute:



(*i*) First, we show existence by constructing $h : \cup_i A_i \stackrel{\cdot}{\to} F$, which we define for every $C \in |\mathbf{C}|$.

$$\begin{array}{rccl} (h)_C : & (\cup_i A_i)(C) & \to & F(C) \\ & x & \mapsto & f_{i_0}(x), \text{ where } i_0 = min\{i; x \in A_i(C)\} \end{array}$$

Note that, $h(in_i(x)) = f_{i_0}(x) = f_i(x)$ because $f_i \circ \alpha_{i_0} = f_{i_0}$.
(*ii*) To show uniqueness, suppose there exists a second map $H : \cup_i A_i \stackrel{\cdot}{\to} F$ which makes the following diagram commute:

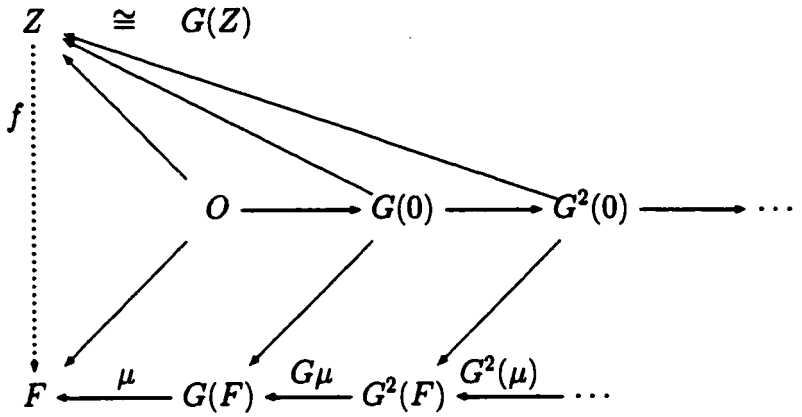


Hence $(H \circ in_j)_C(x) = (f_j)_C(x)$ for all $C \in |GC|$. However $(H \circ in_j)_c(x) = H_C(x) = (f_j)_C(x) = h_C(x)$, hence $H = h$.

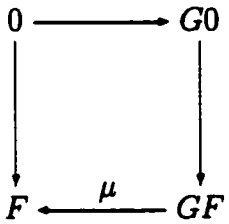
This shows that we have found the colimit of the sequence. □

Theorem 2.7.6 *Let* $G : \mathbf{Set}^{\mathbf{C}} \to \mathbf{Set}^{\mathbf{C}}$ *be an endofunctor of the category* $\mathbf{Set}^{\mathbf{C}}$. *If* Z *is the colimit of the sequence* $0 \to G0 \to G^2 0 \to \ldots$ *and* G *preserves directed unions, then* Z *is the least fixed point of* G.

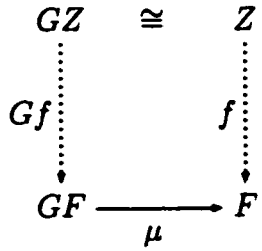
Proof. Since G preserves directed union and by Lemma 2.7.5, the colimit of the given sequence is the directed union of its elements, there is an isomorphism $\eta : G(Z) \to Z$. To have that Z is the least fixed point, we show that (Z, η) is initial in $(G : \mathbf{Set}^{\mathbf{C}})$. Let (F, μ) be an arbitrary element in $(G : \mathbf{Set}^{\mathbf{C}})$,



Note that the square



commutes, since 0 is initial. So all the other squares commute, by applying G^i, $i \geq 0$. So there are maps $G^i(0) \to F$ by composition. Therefore, by the universal property of colimits there exists a unique $f : Z \to F$. Hence we have that the unique map $f : Z \to F$ induces a G-algebra homomorphism from (Z, η) to (F, μ), which makes the following diagram commute.



Commutativity arises from the equation $f = \mu \circ Gf \circ \eta^{-}1$ which is due to the unicity of f. Since (F, μ) was an arbitrary element of the G-algebra category we have that (Z, η) is the initial element of $(G : \mathbf{Set}^{\mathbf{C}})$. Hence, Z is the least fixed point of G. □

Therefore, with the above results of fixed points, we shall be able to construct our object of processes in our model for the finite π-calculus.

Chapter 3

The Finite π-calculus

The following chapter arises from the works by R. Milner, [16, 17], and R. Milner, J. Parrow, and D.Walker, [18], in their construction of a model for concurrent computation. Concurrent computation is a concept which consists of a collection of entities, which we refer to as processes, trying to communicate with one another. The concept of concurrency arises in the ability of the processes to influence each other's and their environment's possible actions. In the π-calculus model for concurrency, the interaction is observed through the passing of messages between respective processes.

The π-calculus is based upon the notion of "naming": we assign names to the links between processes in order to identify which processes may interact. One way of visualizing this is to think of the management of an individual's e-mail addresses. Imagine three individuals with two e-mail accounts each with distinct e-mail addresses. They distribute their personal e-mail addresses in order to be able to communicate with some, or all of the other individuals (they may give more than one per person). The knowledge of an e-mail address is sufficient to enable one individual to communicate with another, therefore we have created a system of possible communications. We will see that the π-calculus is a tool which describes communicating systems in such a way that we can capture the changing structure of the processes in a system. It is a way of describing and analyzing systems consisting of processes which can interact among their neighbors, and whose configurations or environment are continually changing.

This change arises not only from the fact that processes can be arbitrarily linked, but also by a single communication which may carry information that can either create or destroy links between processes. Keeping to our analogy, a change in our system occurs if an individual communicates someone's e-mail account to another individual who did not originally possess it; this creates a new link between the receiver and the owner of the exchanged e-mail address. To destroy an existing link, an individual can cancel one of his e-mail addresses. In doing so, any link to that individual through that e-mail address is forfeited.

The π-calculus focuses on ever-changing situations, i.e. mobility of named links between the processes of a system. From its first presentation in [18], the word "mobility" has become a key point and is much more widely used. This is due to networking technology and the worldwide web, two concepts which depend on the continual changing of links.[1]

In the following section, we will study a much simpler version of the π-calculus: the finite fragment of the π-calculus, i.e. we will not discuss replication processes. We make the simplification since it is this fragment of the π-calculus we are interested to model in Chapter 4. However, we shall see that all the structural properties of the π-calculus are also found for its finite fragment consisting of only the finite processes. The following is based on works by the previously mentioned authors. We shall, henceforth refer to the finite fragment of the π-calculus, as the finite π-calculus.

3.1 The Finite π-calculus

The finite π-calculus is a simplified version of its initial definition in [18]. Although it has fewer process expressions, it can still be used to describe many types of systems. In this section we shall describe the finite π-calculus processes formally, and take an informal look at communication between processes, which we refer to as reaction.

As discussed, "naming" is an important concept in the finite π-calculus. Thus we

[1] A reasoning first brought up by Milner in [16]

begin by identifying the set of all possible "names", with which we can label the links between our processes.

Definition 3.1.1 Assume a set of *names* $\mathcal{N}$, to be the set of links or channels available between processes. They shall be denoted by lower case letters, i.e. $x, y, z, \ldots$ Define a set of *action names*, $Act := \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$, where $\overline{\mathcal{N}} = \{\bar{x}; x \in \mathcal{N}\}$. The set of action names is partitioned such that input action labels are in the set $\mathcal{N}$, output actions labels are in the set $\overline{\mathcal{N}}$, and the silent action label is in the set $\{\tau\}$.

Definition 3.1.2 The set $Proc_f^\pi$ of *Finite π-calculus process expressions* is defined by the following syntax: Let I be a finite indexing set,

$$P \ ::= \ 0 \ | \ P_1|P_2 \ | \ \textstyle\sum_{i\in I} \pi_i.P_i \ | \ \nu z P \ | \ \pi.P$$

where

1. $\pi.P$ represents a *prefixed* process with prefix π:

$$\pi \ ::= \ x(z) \ | \ \overline{x}\langle y\rangle \ | \ \overline{x}(z) \ | \ \tau$$

such that:

- $x(z)$ represents an *input* prefix. It is the action of receiving a name along the channel x. It will subsequently substitute the received name for the placeholder name z throughout the remainder of the process expression.
- $\overline{x}\langle y\rangle$ represents an *output* prefix. It is the action of sending the name y along the channel x.
- $\overline{x}(z)$ represents a *bound output* prefix. It is the action of sending a *local* or *fresh* name, i.e. no other process may contain the name z (except for the process which is prefixed by $\overline{x}(z)$) along the channel x.
- τ represents a *silent action* prefix. It is used to exhibit an internal action in a process, i.e. it represents an action that cannot be modelled with the existing prefixes.

2. 0 represents the *nil* process, i.e. the process which does nothing. It is also considered as the empty sum, i.e. $|I| = 0$.

3. $P_1|P_2$ represents two processes P_1 and P_2 acting in *concurrent communication* or in *parallel*, which allows the processes to act simultaneously with each other or with other processes in their environment.

4. $\sum_{i\in I} \pi_i.P_i$ called *non-deterministic sum* of $\pi_i.P_i$, represents a process able to take part in one, and only one, of the possible alternatives for communication. The choice is not made by the process; it can never commit to one alternative until it occurs, and this occurrence precludes the other alternatives.

5. νzP represents the process P for which we consider the name z to be local, i.e. no other process may contain the name z except the process P. We say that the name z is *restricted* to the process P.

We observe in the definition above that prefixes play a large role in the construction of finite π-calculus processes. This is in fact the reasoning behind the name of this model, π for prefixing, hence we are working in a prefixing calculus. In order to be able to work with these prefixes we define two functions $names, subj$: finite π-calculus prefixes $\rightarrow$ subsets of Act. The first function takes a finite π-calculus prefix and returns all the names with which it is constructed, e.g. $names(x(z)) = \{x, z\}$. On the other hand, the function *subj* takes a finite π-calculus prefix and outputs the *subject* of that prefix, i.e. the link on which it is ready to communicate, e.g. $subj(\overline{x}\langle y\rangle) = \{x\}$. We are now ready to discuss the communication between processes.

The main interest in the finite π-calculus is the possibility of expressing communication between the processes within a system. Thus, we shall need to identify when such communications will be possible. Informally, we understand that communication shall occur when two prefixed processes, one with an input prefix and the other with an output or a bound output prefix such that the subjects of both prefixes are

equal, are placed in concurrent communication. The following definitions will help formalize this concept.

Definition 3.1.3 Let π and π' be finite π-calculus prefixes. If $\pi = x(z)$ and $\pi' \in \{\overline{x}\langle y\rangle, \overline{x}(z')\}$, for any $x, z, y, z' \in \mathcal{N}$, then π and π' are *complementary* prefixes. In other words, π and π' are complementary prefixes if π and π' represent opposite actions acting along the same link.

Definition 3.1.4 In the process expression $\pi.P$, where π is a finite π-calculus prefix, we say that the process P is *guarded* by the prefix π. This implies that the action represented by the prefix π must occur before the process P can become active.

Example 3.1.5 Typical finite π-calculus processes are:

(i) $P_1 = x(u).\overline{u}\langle v\rangle.0 | \overline{x}\langle z\rangle.0$

(ii) $P_2 = x(z).0 + \overline{y}\langle u\rangle.0$

(iii) $P_3 = \tau.(x(z).0)$

(iv) $P_4 = \nu z(\overline{z}(u).\overline{u}\langle y\rangle.0)$

In Example 3.1.5 there are two interesting situations to discuss: the first process $P_1 = x(u).\overline{u}\langle v\rangle.0 | \overline{x}\langle z\rangle.0$ and the last process $P_4 = \nu z(\overline{z}(u).\overline{u}\langle y\rangle.0)$.
P_1 says "receive a (bound) u along x, then send v along the (bound) name u, then do nothing" in parallel with "send z along x then do nothing".
P_4 says "send a (bound) u along (bound) z, then send y along (bound) u, then do nothing" where z is a local name in the whole process due to restriction.
Now we discuss their particularities.

- The process P_1 is the composition of two processes which are guarded by complementary prefixes. We call such a situation a *redex*. The *firing* of a redex, which involves a *reaction* from $P_1 \rightarrow P_1'$, will describe the communication between the two concurrent processes. Our example contains a redex pair: the

prefixes $x(u)$ and $\overline{x}\langle z\rangle$. Then the possible reaction will later be seen to be $P_1 \rightarrow P_1'$, where

$$P_1' = \overline{z}\langle v\rangle.0|0$$

That is in firing the redex, we send z along x to be input for u. This results in the term P_1'. We shall see in Section 3.3.3, the reaction relation that describes all possible firings of such redexes.

- The second process P_4 is the restriction of the name z in the process $P' = \overline{z}(u).\overline{u}\langle y\rangle.0$. In truth, since P' is guarded by the prefix $\overline{z}(u)$, and the name z is fresh for the process P', due to the restriction νz, the process shall never be able to communicate with any process. No process can be guarded by a complementary prefix to $\overline{z}(u)$, since only the process P' has access to the name z because of the restriction. We shall see in Section 3.5, that under a certain relation called *late-bisimulation*, we will consider that P_4 and the process 0 are equivalent processes.

Remark 3.1.6 A similar situation arises when we apply the restriction operator νz, or the prefixes $x(z)$ or $\overline{x}(z)$ with the name z on the process P. Each of these *bind* the name z which appears in its scope, which in this case is the process P. This same binding occurs when we use the quantifiers: $\forall z$ and $\exists z$ in predicate logic, or use the lambda operator: λz in the λ-calculus. In this way, just as with the quantifiers and lambda operator, we create two distinct sets of names within a process: the set of *bound names*, whose elements are the names which are captured within the scope of the above mentioned constructions, and the set of *free names*, whose elements are the names which are not bound. Definition 3.1.7, below describes formally how to construct each set.

Definition 3.1.7 We define two disjoint sets of names in every process P:

1. $\mathbf{fn}(P)$ the *free* names of P, and
2. $\mathbf{bn}(P)$ the *bound* names of P.

which are defined by the following table:

process	free names	bound names
0	$\emptyset$	$\emptyset$
$x(z).P$	$\{x\} \cup (fn(P) - \{z\})$	$\{z\} \cup bn(P)$
$\overline{x}\langle y\rangle.P$	$\{x, y\} \cup fn(P)$	$bn(P)$
$\overline{x}(z).P$	$\{x\} \cup (fn(P) - \{z\})$	$\{z\} \cup bn(P)$
$\nu z P$	$fn(P) - \{z\}$	$bn(P) \cup \{z\}$
$\sum_{i \in I} \pi_i.P_i$	$\cup_{i \in I} fn(\pi_i.P_i)$	$\cup_{i \in I} bn(\pi_i.P_i)$
$P \mid Q$	$fn(P) \cup fn(Q)$	$bn(P) \cup bn(Q)$

Example 3.1.8 Let $P = \nu z(z(z').\nu y(\overline{z'}(y).0) + \overline{z}\langle x\rangle.\overline{x}\langle u\rangle.0)$.
For the process P, we have that $fn(P) = \{x, u\}$ and $bn(P) = \{z, z', y\}$.

Moreover, similarly to both the quantifiers and lambda operator, we do not distinguish between two processes which differ only by a change of bound names. For example, we consider the processes $x(z).0$ and $x(z').0$ as the same process. We shall call two such processes *alpha-convertible*, denoted by $P \equiv_\alpha Q$, to signify that the process P is alpha-convertible to the process Q, by a change of bound variable.

Example 3.1.9 Let $P = \nu z(x(z).\overline{z}\langle y\rangle.0 + \overline{z}(y).y(w).0)$ and $P' = \nu a(x(b).\overline{b}\langle y\rangle.0 + \overline{a}(c).c(w).0)$, then $P \equiv_\alpha P'$:

$$\begin{aligned}
P &= \nu z(x(z).\overline{z}\langle y\rangle.0 + \overline{z}(y).y(w).0) \\
\text{alpha-conv. on bound } z \text{ in scope of } x(z) &\equiv_\alpha \nu z(x(b).\overline{b}\langle y\rangle.0 + \overline{z}(y).y(w).0) \\
\text{alpha-conv. on bound } z \text{ in scope of } \nu z &\equiv_\alpha \nu a(x(b).\overline{b}\langle y\rangle.0 + \overline{a}(y).y(w).0) \\
\text{alpha-conv. on bound } y \text{ in scope of } \overline{a}(y) &\equiv_\alpha \nu a(x(b).\overline{b}\langle y\rangle.0 + \overline{a}(c).c(w).0) \\
&= P'
\end{aligned}$$

It will be necessary to be able to substitute one free name in a process for another, as we described in Definition 3.1.2 when describing the input prefixed process. Moreover, we have described communication as an exchange of names, whereas the received name substitutes for an existing one in the process. To enable us to define substitution, we shall proceed by structural induction on the set of processes.

Definition 3.1.10 Define a new process expression $P[z := y]$, called the *substitution* of z for y in the process P. It represents the process P with every free occurrence of the name z replaced by the name y, changing the bound names in P through alpha-conversion if the need arises. This change of bound names is done to avoid the name y from becoming captured in the scope of a νy, $x(y)$, or $\overline{x}(y)$, (which would change the occurrence of the free name z in P to a bound name y).

Remark 3.1.11 It is important that when substituting a free name y, that it does not become bound in the scope of any of the prefixes $x(y)$ or $\overline{x}(y)$, or by the restriction νy. Such capture would change the whole meaning of the process, as we can see in the following example.

Example 3.1.12 Let $P = \nu z(x(z).\overline{z}\langle y\rangle.0 + \overline{z'}\langle z\rangle.z(z').\overline{z'}\langle y\rangle.0)|\overline{z}(y).\overline{x}\langle y\rangle.0$. Note that $fn(P) = \{x, y, z', z\}$ and $bn(P) = \{z, z', y\}$, and that these two sets are not disjoint. However, observe that we can make them disjoint by using alpha-conversion. We wish to compute $P[y := u]$, $P[z' := v]$, and $P[z := s]$. The first step is to change our process P to a process P' such that $P \equiv_\alpha P'$ and all the bound names are fresh in our process, i.e. there are no identical free names and bound names in the process P.

Set $P' = \nu a(x(b).\overline{b}\langle y\rangle.0 + \overline{z'}\langle a\rangle.a(c).\overline{c}\langle y\rangle.0)|\overline{z}(d).\overline{x}\langle d\rangle.0$, where $P \equiv_\alpha P'$. Notice that $fn(P') = \{x, y, z', z\}$ and $bn(P') = \{a, b, c, d\}$. Now we can make our substitutions:

- $P[y := u] \equiv_\alpha \nu a(x(b).\overline{b}\langle u\rangle.0 + \overline{z'}\langle a\rangle.a(c).\overline{c}\langle u\rangle.0)|\overline{z}(d).\overline{x}\langle d\rangle.0$
- $P[z' := v] \equiv_\alpha \nu a(x(b).\overline{b}\langle y\rangle.0 + \overline{v}\langle a\rangle.a(c).\overline{c}\langle y\rangle.0)|\overline{z}(d).\overline{x}\langle d\rangle.0$
- $P[z := s] \equiv_\alpha \nu a(x(b).\overline{b}\langle y\rangle.0 + \overline{z'}\langle a\rangle.a(c).\overline{c}\langle y\rangle.0)|\overline{s}(d).\overline{x}\langle d\rangle.0$

However without having done our initial alpha-conversion it would be easy to make the following mistakes:

- $P[z' := v] = \nu z(x(z).\overline{z}\langle y\rangle.0 + \overline{v}\langle z\rangle.z(z').\underline{\overline{v}\langle y\rangle}.0)|\overline{z}(y).\overline{x}\langle d\rangle.0$, we see that we have substituted for the bound z' in the underlined prefix, so the bound z' becomes the free variable v.

- $P[x := y] = \nu z(y(z).\overline{z}\langle y\rangle.0 + \overline{z'}\langle s\rangle.z(z').\overline{z'}\langle y\rangle.0)|\overline{z}(y).\underline{\overline{y}\langle y\rangle}.0$, we have misused our substitution once again. Notice that the free name x has been substituted by the name y in the underlined prefix. However, the name y is now bound. This is not allowed by the definition of our substitutions, and we must be wary of this occurrence.

3.2 A Working Example: Private Key Generation

To help illustrate a small part of what type of systems the finite π-calculus can describe, we will see an example for which mobility plays an important role.

Example 3.2.1 Consider a simplified authentication protocol between two parties which depends on a trusted third party. The protocol is very simple and is described as follows:

- A initiates contact with the S by sending the name of their shared link along with the shared link between B and S.
- S sends to A and B the name of a fresh link.
- A and B use the received name to communicate.

We can describe this system in the finite π-calculus by creating a system of three interacting processes: A, B, and S.

Consider the process A in Figure 1. Notice that it has two channels along which it can communicate. We construct A by analyzing the actions that it must perform:

- It initiates a demand for a fresh link with which to communicate with B by sending the names a and b along the *request* channel. Then it listens along a for the private channel, then sends his message link m to B through the received channel. This can be written as

$$\overline{request}\langle a, b\rangle.a(private).\overline{private}\langle m\rangle.0$$

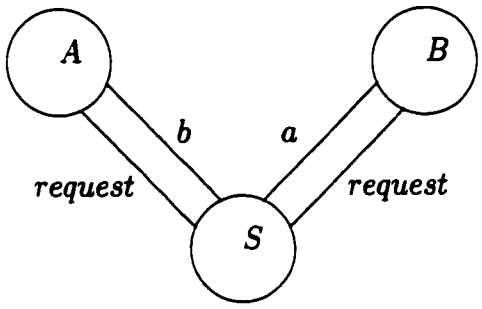


Figure 1: Simplified Protocol

- It listens for a private channel along its shared link with S, then listens for a message along that link. This can be written as

$$a'(private).private(x).0$$

Hence we can describe the full behavior of A as follows

$$A = \overline{request}\langle a, b\rangle.a(private).\overline{private}\langle m\rangle.0 + a'(private).private(x).0$$

Similarly, we define the process which represents B as

$$B = \overline{request}\langle b', a'\rangle.b'(private).\overline{private}\langle m'\rangle.0 + b(private).private(x).0$$

Finally, S is a process which must listen along the *request* channel for two links to which to send a fresh name *priv* to the received links. This can be written as

$$S = \nu priv(request(sender, receiver).\overline{sender}\langle priv\rangle.\overline{receiver}\langle priv\rangle.0)$$

We describe the whole system by placing our processes in concurrent communication as follows:

$$System = (A|B|S)$$

After we have seen the reaction relation we shall be able to verify that

$$System \rightarrow^* 0$$

where $\rightarrow^*$ represents the transitive closure of the reaction relation $\rightarrow$, seen in Section 3.3.3. It implies that after a number of communications *System* will act as 0.

3.3 Structural Congruence and Reaction Relation

One important structural property of the finite π-calculus is the reaction relation. This relation describes all possible communications between concurrent processes. However, before defining the reaction relation one needs to specify the notion of an equivalence between processes.

3.3.1 Process Congruence

We must formalize the notion of "congruence", i.e. behavioral equality between processes, since we will define several congruence relations of processes throughout this chapter. Informally, we should consider two processes P and Q to be congruent if they have equivalent behaviors. However, what do we consider equivalent behavior? It is not unreasonable to consider that if a process P is interchangeable with any possible instance of the process Q and behaves exactly like Q in that instance, then P and Q are congruent. More formally, we have:

Definition 3.3.1 A *process context* C is of the form:

$$C \ ::= \ [-] \ | \ \pi.C + M \ | \ C|P \ | \ P|C \ | \ \nu z C$$

where M is a process of the form $\sum_{i\in I} \pi_i.P_i$, with I a finite indexing set ($|I| = 0$ implies $\sum_{i\in I} \pi_i.P_i = 0$), and $P_i, P \in Proc_f^\pi$. $C[Q]$ represents the process context C in which we have filled the "hole" with the process Q. The *elementary contexts* are:

$$\pi.[-] + M, \ \ [-]|P, \ \ P|[-], \ \ \nu z[-]$$

Remark 3.3.2 A process context is not a finite π-calculus process because of the "hole" which has no meaning as a finite π-calculus process. This "hole" represents an empty slot in the process which is ready to be filled by any finite π-calculus process expression. Moreover, when filling a "hole" in a process context we do not have the same restriction as when we substitute names in a process. When we compute $P[z := x]$, we must only replace the free occurrences of the name z by x in the process P; however when we compute $C[Q]$ we do not have such restrictions. We simply replace the "hole" by the process Q, and let free names of Q become bound if they become captured.

Now, based upon the notion of process contexts, we can define what an equivalence relation between processes must satisfy to be considered a process congruence.

Definition 3.3.3 Let $\cong$ be an equivalence relation over $Proc_f^\pi$. Then $\cong$ is said to be a *process congruence* if it is preserved by all elementary contexts; that is, if $P \cong Q$ then

$$\begin{array}{rcl} \pi.P + M & \cong & \pi.Q + M \\ \nu z P & \cong & \nu z Q \\ P|R & \cong & Q|R \\ R|P & \cong & R|Q \end{array}$$

Example 3.3.4 The equivalence relation $\equiv_\alpha$ defined earlier such that $P \equiv_\alpha Q$ implies that P and Q are alpha-convertible, forms a process congruence.

3.3.2 Structural Congruence

In the definition below we shall construct another process congruence by expanding our definition of the $\equiv_\alpha$ relation to encompass more than simple alpha-conversion.

Definition 3.3.5 Two process expressions P and Q in the finite π-calculus are *structurally congruent*, written $P \equiv Q$, if we can transform one into the other by using the following equations (in either direction):

1. If $P \equiv_\alpha Q$, then $P \equiv Q$. In other words, $\equiv_\alpha \subset \equiv$.

2. $P|Q \equiv Q|P$, $P|0 \equiv P$, and $(P|Q)|R \equiv P|(Q|R)$

3. $P+Q \equiv Q+P$, $P+0 \equiv P$, and $(P+Q)+R \equiv P+(Q+R)$

4. $\nu z 0 \equiv 0$, $\nu z \nu z' P \equiv \nu z' \nu z P$, and $\nu z(P|Q) \equiv P|(\nu z Q)$, if $z \notin fn(P)$

Remark 3.3.6 Once again the similarity between the restriction operator and quantifiers are made obvious here by the equations it must satisfy. In predicate logic, we have that the existential quantifier must satisfy a similar condition as the restriction operator, i.e. $\exists x(A \vee B(x)) = A \vee \exists x B(x)$ if x is not free in the formula A. In [15], the authors define the concept of hyperdoctrines, a fibred categorical model for quantified logic, as a model of concurrent constraint logic programming. This paradigm is similar to finite π-calculus in that it has hiding (restriction) and parallel composition. It is shown there that hiding satisfies the same categorical properties as an existential quantifier. This proof does not directly apply to the finite π-calculus here, but seems to apply to a variant of the π-calculus, the π_I-calculus, seen in [22].

Proposition 3.3.7 *Structural congruence $\equiv$ is a process congruence over $Proc_f^\pi$.*

Hence, we have constructed a process congruence which allows us to ascertain if two processes are equivalent. Moreover, we shall use structural congruence in our definition of the reaction relation below, as one of the inference rules.

3.3.3 Reaction Relation

Communication between two processes P and Q will occur, if they are in concurrent communication and they are guarded by complementary prefixes. The two main situations are:

- The complementary prefixes are input $P = x(z).P'$ and output $Q = \overline{x}\langle y\rangle.Q'$. The reaction consists of many simultaneous actions. First, Q sends the name y along the link x, and P receives the name y along the same link. After Q's prefix has finished its action the process continues as Q', while P's prefix has also finished its action and the process continues as $P'[z := y]$, i.e. with the free

occurrences of the placeholder name z replaced by the received name y. Finally, after communication the continuations of the processes remain in parallel, hence we get $P'[z := y]|Q'$.

- The complementary prefixes are input $P = x(z).P'$ and bound output $\overline{x}(z).Q'$. The reaction consists once more of many simultaneous actions. First, Q sends the fresh name z along the channel x, while P is receiving the fresh name z along the channel x. However, since the name z is fresh the end process must also express that the received name is fresh. We restrict the name z on the continuation of both the process P and Q. Hence we get $\nu z(P'|Q')$.

This idea is formalized in the notion of "reaction", axiomatized by the inference rules below.

Definition 3.3.8 The *reaction relation* $\rightarrow$ over $Proc_f^\pi$ contains exactly those transitions which can be inferred from the rules below: Let $P \in Proc_f^\pi$ and M, N be summation processes,

TAU: $$\tau.P + M \rightarrow P$$

REACT1[x]: $$(x(z).P + M)|(\overline{x}(y).Q + N) \longrightarrow P[z := y]|Q$$

REACT2[x]: $$(x(z).P + M)|(\overline{x}(y).Q + N) \longrightarrow \nu z(P|Q)$$

PAR: $$\frac{P \rightarrow P'}{P|Q \rightarrow P'|Q}$$

RES: $$\frac{P \rightarrow P'}{\nu z P \rightarrow \nu z P'}$$

STRUCT: $$\frac{P \rightarrow P'}{Q \rightarrow Q'} \quad \text{if } P \equiv Q \text{ and } P' \equiv Q'$$

Example 3.3.9 From the discussion ensuing from Example 3.1.5 we had stated that $P_1 \rightarrow P_1'$, where $P_1 = x(u).\overline{u}\langle v\rangle.0|\overline{x}\langle z\rangle.0$ and $P_1' = \overline{z}\langle v\rangle.0|0$. From the reaction relation above we see that the reaction is due to the $REACT1[x]$ rule.

Moreover, we shall define an extension of the reaction relation $\rightarrow$, denoted $\rightarrow^*$, such that $P \rightarrow^* P'$ will mean that there exists a finite number of processes $P = P_1, P_2, \ldots, P_n = P'$ such that $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow \ldots \rightarrow P_n$. Hence, $\rightarrow^*$ is the transitive closure of the reaction relation. Now we are ready to check our claim from Example 3.2.1

Example 3.3.10 Now we can see how in the mobile phone example, how $System \rightarrow^* 0$:

$$
\begin{aligned}
System &= (\overline{request}\langle a, b\rangle.a(private).\overline{private}\langle m\rangle.0 + a'(private).private(x).0) \mid \\
&\quad (\overline{request}\langle b', a'\rangle.b'(private).\overline{private}\langle m'\rangle.0 + b(private).private(x).0) \mid \\
&\quad (\nu priv(request(sender, receiver).\overline{sender}\langle priv\rangle.\overline{receiver}\langle priv\rangle.0)) \\
&\rightarrow \nu priv(a(private).\overline{private}\langle m\rangle.0 + a'(private).private(x).0) \mid \\
&\quad (\overline{request}\langle b', a'\rangle.b'(private).\overline{private}\langle m'\rangle.0 + b(private).private(x).0) \mid \\
&\quad (\overline{a}\langle priv\rangle.\overline{b}\langle priv\rangle.0)) \\
&\rightarrow \nu priv(\overline{priv}\langle m\rangle.0 \mid (\overline{request}\langle b', a'\rangle.b'(private).\overline{private}\langle m'\rangle.0 \\
&\quad + b(private).private(x).0) \mid (\overline{b}\langle priv\rangle.0)) \\
&\rightarrow \nu priv(\overline{priv}\langle m\rangle.0 \mid priv(x).0) \mid 0) \\
&\rightarrow \nu priv(0 \mid 0 \mid 0) \\
&\equiv 0
\end{aligned}
$$

3.4 The Finite π-calculus as a Labelled Transition System

We have previously described a process congruence on the processes of the finite π-calculus, i.e. the structural congruence. However, structural congruence does not permit all the obvious equivalences acceptable by our intuition.

Example 3.4.1 If $x \neq y$, the processes $x(z).0|\overline{y}\langle u\rangle.0$ and $x(z).\overline{y}\langle u\rangle.0 + \overline{y}\langle u\rangle.x(z).0$ should be considered to be equivalent, since they behave equivalently with their environments. The behavior of both processes follow the same pattern. There are two possible cases:

- It could receive a name z through the link x, then send the name u along the link y, and then act as the process 0.
- It could send the name u along the link y, then receive a name z along the link x, then act as the process 0.

In either case both processes can do exactly the same actions the other process can. We shall formalize this into an equivalence relation in Section 3.4.2.

We will define an equivalence relation called late-bisimulation based on the behavior of processes. To understand how to describe the behaviors of processes we shall describe the finite π-calculus as a labelled transition system. In this section we shall formally define labelled transition systems and describe how we construct a labelled transition system from the finite π-calculus.

3.4.1 Labelled Transition Systems

Definition 3.4.2 A *labelled transition system (= LTS)* over a set of names $\mathcal{N}$ is a triple $(\mathcal{Q}, L, \mathcal{T})$ consisting of

- a set $\mathcal{Q}$ of *states*;
- a set L of *labels*;
- a ternary relation $\mathcal{T} \subseteq (\mathcal{Q} \times L \times \mathcal{Q})$, known as a *transition relation*

If $(q, \alpha, q') \in \mathcal{T}$ we write $q \xrightarrow{\alpha} q'$, and we call q the *source* and q' the *target* of the transition. If $q \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} q_n$ then we call q_n a *derivative* of q under $\alpha_1\alpha_2\cdots\alpha_3$.

To see that it will be a natural step to view the finite π-calculus as an LTS, here is an example of an LTS, adapted from an example from [16].

Example 3.4.3 A typical example of an LTS is a vending machine. Let's create a vending machine which sells Coke in small cans or large bottles, where a small can is \$1.00 and a large bottle is \$2.00. There are three possible actions: insert \$1.00 (I), press the can button (C) and press the bottle button (B). We can see the system in Figure 2.

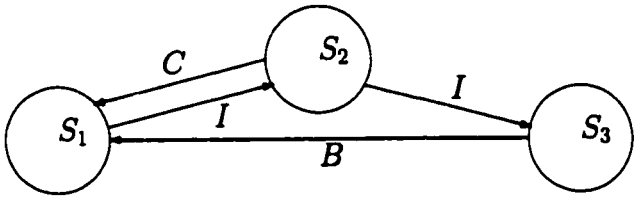


Figure 2: Vending Machine

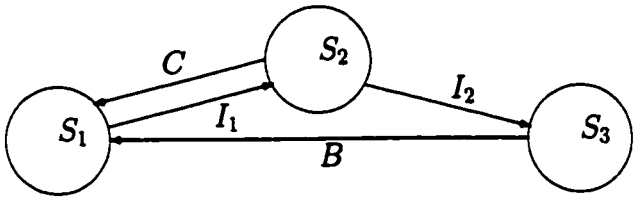


Figure 3: Modified Vending Machine

This structure can also be interpreted in a finite π-calculus system. The states are represented by processes and the transitions by links, where the source of the arrow is sending along the link and the target is receiving along the link. However, since we wish to distinguish between the two actions of inserting \$1.00, we represent the two actions with a summand, i.e. I_1 and I_2, as seen in Figure 3. This is done to avoid the possibility of getting a bottle for the price of a can. The system can be represented with the following processes:

$$S_1 = (\overline{I_1}(x).(C(y).\tau.0 + B(y).\tau.0) \quad (4)$$

The process S_1 can be described as doing the following:

- It sends a signal along the channel I_1, signifying that someone has inserted \$1.00, then it can do one of two things:

  (i) It waits for a signal along the C channel, i.e. it waits for someone to press the "can" button, then it will give out a can of Coke, represented by the internal action τ.

  (ii) It waits for a signal along the B channel, i.e. it waits for someone to press the "bottle" button, then it will give out a bottle of Coke, represented by the internal action τ.

$$S_2 = (I_1(y).(\overline{I_2}\langle x\rangle.0 + \overline{C}\langle x\rangle.0) \quad (5)$$

The process S_2 can be described as doing the following:

- It can listen for a signal along the I_1 channel, i.e. it waits for someone to insert \$1.00, then it can do one of two things:

  (i) It sends a signal along the channel I_2, signifying that someone has inserted the second \$1.00, then becomes the nil process.

  (ii) It sends a signal along the channel C, signifying that someone has pressed the "can" button, then it becomes the nil process.

$$S_3 = (I_2(y).\overline{B}\langle x\rangle.0) \quad (6)$$

The process S_3 can be described as doing the following:

- It listens for a signal along the channel I_2, i.e. it waits for someone to insert the second \$1.00, then it sends a signal along the channel B, signifying someone has pressed the "bottle" button, then it becomes the nil process.

Now we compose the processes together and restrict all the names in order to guarantee only this system has access to these labels.

$$\begin{aligned} Vend &= \nu I_1 \nu I_2 \nu C \nu B(S_1|S_2|S_3) \\ &= \nu I_1 \nu I_2 \nu C \nu B((\overline{I_1}\langle x\rangle.(C(y).\tau.0 + B(y).\tau.0))| \\ &\quad (I_1(y).(\overline{I_2}\langle x\rangle.0 + \overline{C}\langle x\rangle.0))|(I_2(y).\overline{B}\langle x\rangle.0)) \end{aligned}$$

3.4.2 Commitment

As we have seen in Example 3.4.3 we can express finite π-calculus systems as an LTS. In this section we construct an LTS of finite π-calculus processes $(\mathcal{Q}, L, \mathcal{T})$ where L is the set of all finite π-calculus prefixes, $\mathcal{Q}$ is the set of all processes in $Proc_f^\pi$ and $\mathcal{T}$ is the commitment relation seen below. The following is adapted from Milner [17].

Definition 3.4.4 We shall define *commitment* to be the relation generated by the following axioms and rules of inference: Let π be a finite π-calculus prefix, $P, Q \in Proc_f^\pi$ and M a summation process,

PRE $\quad \pi.P \xrightarrow{\pi} P$

SUM1 $\quad \dfrac{P \xrightarrow{\pi} P'}{P + M \xrightarrow{\pi} P'}$ $\qquad$ SUM2 $\quad \dfrac{P \xrightarrow{\pi} P'}{M + P \xrightarrow{\pi} P'}$

PAR1 $\quad \dfrac{P \xrightarrow{\pi} P'}{P|Q \xrightarrow{\pi} P'|Q}$ $\qquad$ PAR2 $\quad \dfrac{P \xrightarrow{\pi} P'}{Q|P \xrightarrow{\pi} Q|P'}$

COM1 $\quad \dfrac{P \xrightarrow{x(z)} P' \quad Q \xrightarrow{\overline{x}\langle y\rangle} Q'}{P|Q \xrightarrow{\tau} P'[z := y]|Q'}$ $\qquad$ COM2 $\quad \dfrac{P \xrightarrow{x(z)} P' \quad Q \xrightarrow{\overline{x}\langle y\rangle} Q'}{Q|P \xrightarrow{\tau} Q'|P'[z := y]}$

CLOSE1 $\quad \dfrac{P \xrightarrow{x(z)} P' \quad Q \xrightarrow{\overline{x}(z)} Q'}{P|Q \xrightarrow{\tau} \nu z(P'|Q')}$ $\qquad$ CLOSE2 $\quad \dfrac{P \xrightarrow{x(z)} P' \quad Q \xrightarrow{\overline{x}(z)} Q'}{Q|P \xrightarrow{\tau} \nu z(Q'|P')}$

OPEN $\quad \dfrac{P \xrightarrow{\overline{x}(z)} P'}{\nu z P \xrightarrow{\overline{x}(z)} P'}$ $\qquad$ RES $\quad \dfrac{P \xrightarrow{\pi} P'}{\nu z P \xrightarrow{\pi} \nu z P'}\ \pi \notin names(\pi)$

Example 3.4.5 We can now see how our two processes used in the Example 3.4.1 have similar behavior by listing their possible commitments. Let $x \neq y$,

- All the possible commitments of the process $x(z).0|\overline{y}\langle u\rangle.0$ are:

$$x(z).0|\overline{y}\langle u\rangle.0 \begin{cases} \overset{x(z)}{\rightarrow} & 0|\overline{y}\langle u\rangle.0 \\ \overset{\overline{y}\langle u\rangle}{\rightarrow} & x(z).0|0 \end{cases} \quad \begin{cases} \overset{\overline{y}\langle u\rangle}{\rightarrow} & 0|0 \\ \overset{x(z)}{\rightarrow} & 0|0 \end{cases}$$

- All the possible commitments of the process $x(z).\overline{y}\langle u\rangle.0 + \overline{y}\langle u\rangle.x(z).0$ are:

$$x(z).\overline{y}\langle u\rangle.0 + \overline{y}\langle u\rangle.x(z).0 \begin{cases} \overset{x(z)}{\rightarrow} & \overline{y}\langle u\rangle.0 \\ \overset{\overline{y}\langle u\rangle}{\rightarrow} & x(z).0 \end{cases} \quad \begin{cases} \overset{\overline{y}\langle u\rangle}{\rightarrow} & 0 \\ \overset{x(z)}{\rightarrow} & 0 \end{cases}$$

Finally, since $0|0 \equiv 0$, $0|\overline{y}\langle u\rangle.0 \equiv \overline{y}\langle u\rangle.0$ and $x(z).0|0 \equiv x(z).0$ we have that no matter which commitment is chosen for each process the other process can match it.

There is a relation between the commitment relation and the reaction relation. Notice that a τ transition from the commitment relation (i.e. COM, CLOSE rules) mimics very closely the REACT rules from the reaction relation. The following few results from Milner [16], demonstrate that the reaction relation is equivalent to a τ transition up to structural congruence.

Lemma 3.4.6 *If $P \rightarrow P'$, by the reaction relation, then $P \xrightarrow{\tau} \equiv P'$, where we define the relation $P \xrightarrow{\tau} \equiv P'$ to signify that there exists a P'' such that $P \xrightarrow{\tau} P''$ and $P'' \equiv P'$.*

Lemma 3.4.7 *Let $P \xrightarrow{\pi} P'$. Then P and P' can be expressed, up to structural congruence, in the form*

$$\begin{aligned} P &\equiv \nu\vec{z}((\pi.Q + M)|R) \\ P' &\equiv \nu\vec{z}(Q|R) \end{aligned}$$

where π is not captured by $\nu\vec{z}$, M is a summation of processes, and Q, R are processes.

Which allow us to prove the equivalence theorem.

Theorem 3.4.8 *$P \xrightarrow{\tau} \equiv P'$ if and only if $P \rightarrow P'$.*

Therefore, the above sequence of results allows us to prove a close relationship between the reaction relation $\rightarrow$ and a τ commitment $\xrightarrow{\tau}$ in the commitment relation. We end this section by stating some useful facts about the commitment relation transitions, due to Milner [16].

Proposition 3.4.9
(i) Given P, there are only finitely many transitions $P \xrightarrow{\pi} P'$.
(ii) If $P \xrightarrow{\pi} P'$ then $fn(P', \pi) \subseteq fn(P)$, where $fn(P', \pi) = fn(P') \cup fn(\pi.0)$.
(iii) If $P \xrightarrow{\pi} P'$ and σ is any substitution then $P\sigma \xrightarrow{\pi\sigma} P'\sigma$.

3.5 Late-bisimulation

We now introduce the notion of *late-simulation* and *late-bisimulation* between processes. From these concepts we can create an equivalence relation on the states of an LTS. In particular, they will define equivalence relations on the LTS of finite π-calculus processes. The late-bisimulation relation will be able to differentiate between processes which have different behaviors, such as the possibility of non-determinism.

Example 3.5.1 A classic example seen in [16] is the use of simulation to differentiate between two similar vending machines A and B, see Figure 4.

Notice that both vending machines have buttons a, b and c; however in machine B, the activation of the button a leaves us with one choice of action either b or c but never both. Hence the vending machine B's first move decides the choice of our second action, unlike the case of machine A. Thus, we should have an equivalence relation which differentiates between these two machines.

With the following notion of late-simulation we shall be able to distinguish between the two vending machines.

3.5.1 Late-Simulation

Definition 3.5.2 Let $(Proc_f^\pi, Prefixes, Commitment)$ be an LTS for the finite π-calculus, and let $\mathcal{S}$ be a binary relation over $Proc_f^\pi$. Then $\mathcal{S}$ is called a *late-simulation* over $(Proc_f^\pi, Prefixes, Commitment)$ if it satisfies the following: If $P\mathcal{S}Q$ implies that

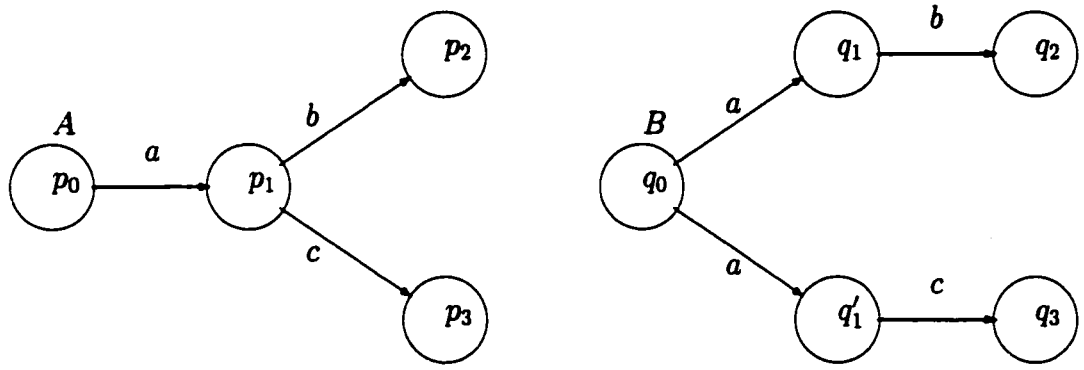


Figure 4: Two vending machines, A and B

1. If $P \xrightarrow{x(z)} P'$ and $z \notin fn(Q)$, then for some Q', $Q \xrightarrow{x(z)} Q'$ and for all names c, $P'[z := c]\mathcal{S}Q'[z := c]$.
2. If $P \xrightarrow{\alpha} P'$ for $\alpha \in \{\overline{x}\langle y\rangle, \tau\}$, then Q' exists s.t. $Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{S}Q'$.
3. If $P \xrightarrow{\overline{x}(z)} P'$ and $z \notin fn(Q)$, then Q' exists s.t. $Q \xrightarrow{\overline{x}(z)} Q'$ and $P'\mathcal{S}Q'$

We say that Q *late-simulates* P if there exists a late-simulation S such that PSQ.

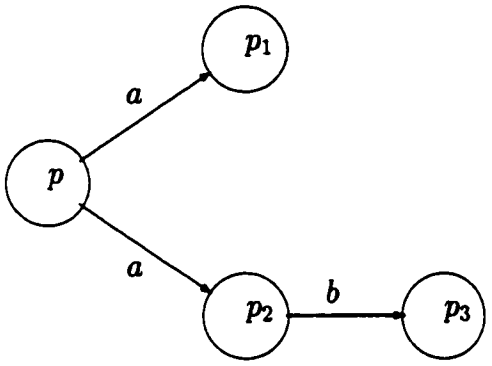
Example 3.5.3 If we return to our vending machines of Example 3.5.1, we can find a late-simulation $\mathcal{S}$ such that p_0 late-simulates q_0.

$$\mathcal{S} = \{(q_0, p_0), (q_1, p_1), (q_1', p_1), (q_2, p_2), (q_3, p_3)\}$$

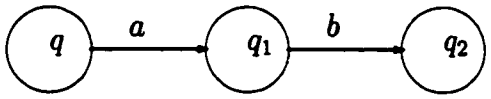
However, it will be impossible to have a relation $\mathcal{S}'$ such that q_0 late-simulates p_0, since any such relation must contain the pair (p_0, q_0) which has but one transition $p_0 \xrightarrow{a} p_1$. Here the two pairs that are possible are (p_1, q_1) or (p_1, q_1'), but p_1 does not strongly simulate q_1 nor q_1'.

To have an equivalence relation based on late-simulation, we will need a stronger condition than: p is late-bisimular to q if and only if p late-simulates q and q late-simulates p. We can see why in the following example, adapted from [16].

Example 3.5.4 Consider the two systems proposed as an exercise in [16]:



System A



System B

We construct a late-simulation $\mathcal{S}$ such that p late-simulates q and a late-simulation $\mathcal{S}'$ such that q late-simulates p. The relations are:

$$\begin{aligned} \mathcal{S} &= \{(p,q),(p_1,q_1),(p_2,q_1),(p_3,q_2)\} \\ \mathcal{S}' &= \{(q,p),(q_1,p_2),(q_2,p_3)\} \end{aligned}$$

Notice that the above systems are such that we do not want them to be equivalent, since A contains a deadlock i.e. p_1 has no continuation, yet q_1 always has a second action available.

This is why we define late-bisimulation as follows:

Definition 3.5.5 A binary relation S over $\mathcal{Q}$ is said to be a *late-bisimulation* over the LTS $(\mathcal{Q}, L, \mathcal{T})$ if both $\mathcal{S}$ and its converse are late-simulations. We say that P and Q are *late-bisimular*, written $P \sim_l Q$, if there exists a late-bisimulation $\mathcal{S}$ such that $P\mathcal{S}Q$

Example 3.5.6 Hence we can define a late-bisimulation for the following processes:

1. Recall the process from Example 3.1.5, $\nu z(\overline{z}(u).\overline{u}\langle y\rangle.0)$ and 0, with $\mathcal{S}_1 = \{(\nu z(\overline{z}(u).\overline{u}\langle y\rangle.0, 0)\}$, since 0 has no commitments this clearly shows that $\mathcal{S}_1^{-1}$ is a late-simulation and since by the commitment relation we can find no transitions for $\nu z(\overline{z}(u).\overline{u}\langle y\rangle.0)$, $\mathcal{S}_1$ is a late-simulation. Hence, $\nu z(\overline{z}(u).\overline{u}\langle y\rangle.0) \sim_l 0$.

2. An interesting example from [16] shows two processes which are late-bisimular but name substitution will not preserve their late-bisimularity. Notice that $\overline{x}\langle u\rangle.0|y(v).0$ and $\overline{x}\langle u\rangle.y(v).0 + y(v).\overline{x}\langle u\rangle.0$ are late-bisimular under

$$\begin{aligned}\mathcal{S}_2 \;=\; & \{(\overline{x}\langle u\rangle.0|y(v).0, \overline{x}\langle u\rangle.y(v).0 + y(v).\overline{x}\langle u\rangle.0), (0|y(v).0, y(v).0),\\ & (0,0), (\overline{x}\langle u\rangle.0|0, \overline{x}\langle u\rangle.0)\}\end{aligned}$$

since the only commitments of $\overline{x}\langle u\rangle.0|y(v).0$ are $\overline{x}\langle u\rangle.0|y(v).0 \xrightarrow{\overline{x}\langle u\rangle} 0|y(v).0$ and $\overline{x}\langle u\rangle.0|y(v).0 \xrightarrow{y(v)} \overline{x}\langle u\rangle.0|0$, and the other process has the commitments $\overline{x}\langle u\rangle.y(v).0 + y(v).\overline{x}\langle u\rangle.0 \xrightarrow{\overline{x}\langle u\rangle} y(v).0$ and $\overline{x}\langle u\rangle.y(v).0 + y(v).\overline{x}\langle u\rangle.0 \xrightarrow{y(v)} \overline{x}\langle u\rangle.0$. We notice that these are identical commitments. However, if we substitute y for x in both processes we destroy the late-bisimulation between the processes, i.e. $\overline{x}\langle u\rangle.0|x(v).0 \not\sim_l \overline{x}\langle u\rangle.x(v).0 + x(v).\overline{x}\langle u\rangle.0$. Here we can find a commitment $\overline{x}\langle u\rangle.0|x(v).0 \xrightarrow{\tau} 0$ which cannot be matched by the process $\overline{x}\langle u\rangle.x(v).0 + x(v).\overline{x}\langle u\rangle.0$.

As explained in [16], the above example shows that under late-bisimulation we can have $P \sim_l Q$ but $x(z).P \not\sim_l x(z).Q$. Therefore, there is a process context, i.e in this case $x(z).[-]$ which does not preserve late-bisimulation. We construct a process congruence based on late-bisimulation by considering it under all substitution of names. This creates a process congruence called *late-equivalence.*

Definition 3.5.7 Two processes P, Q are *late-equivalent*, written $P \dot{\sim}_l Q$, if $P\sigma \sim_l Q\sigma$ for all substitutions σ.

3.5.2 Late-bisimulation vs Early-bisimulation

Our definition of late-bisimulation is not the only possible equivalence relation we can construct which is based on behavior. It is also possible to alter our definition of the late-bisimulation, in order to create an alternate equivalence relation by simply exchanging the order of the quantifiers in the first property in the late-bisimulation. The new relation is called *early-bisimulation.*

Definition 3.5.8 A relation $\mathcal{S}'$ on processes is an *early-bisimulation* if $P\mathcal{S}'Q$ implies

1. Whenever $P \xrightarrow{x(z)} P'$ and $z \notin fn(Q)$, then for all names c, there is Q' such that $Q \xrightarrow{x(z)} Q'$ and $P'[z := c]\mathcal{S}'Q'[z := c]$.

2. Whenever $P \xrightarrow{\alpha} P'$ for $\alpha \in \{\overline{x}\langle y\rangle, \tau\}$, then Q' exists s.t. $Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{S}'Q'$.

3. Whenever $P \xrightarrow{\overline{x}(z)} P'$ and $z \notin fn(Q)$, then Q' exists s.t. $Q \xrightarrow{\overline{x}(z)} Q'$ and $P'\mathcal{S}'Q'$

4. $(\mathcal{S}')^{-1}$ satisfies 1. - 3.

Processes P and Q are *early-bisimular*, written $P \sim_e Q$, if $P\mathcal{S}'Q$, for some early-bisimulation $\mathcal{S}'$.

Remark 3.5.9 As [18] states, notice that it is clause 1. of each definition of early and late-bisimulation which differ. This makes early-bisimulation weaker than late-bisimulation. That is, more agents are equivalent under the early-bisimulation. The reason is that clause 1 of the late-bisimulation requires that there be one late-simulating input transition which applies for *all* instances of the object. In contrast, clause 1 of the early-bisimulation only requires that for each instance of the object there exist a late-simulating transition (and these late-simulating transitions may be different for different instances). Therefore we may think of $\sim_e$ as happening simultaneously with (or even before) the input transition (i.e "early"), and whereas $\sim_l$ the instantiations may be regarded as happening after the transition (i.e. "late").

3.5.3 Strong Ground Equivalence (SGE) Axiomatization

As we have seen, late-bisimulation is not preserved under substitution. For this reason we often refer to late-bisimulation as *strong ground equivalence*. In this section we define an axiomatic theory for late-bisimulation, i.e. we create an axiomatic list of late-bisimular processes in the finite π-calculus with inference rules. These axioms and inference rules will generate all the possible late-bisimular pairs in the finite π-calculus.

To understand the intricacies of the late-bisimulation of the parallel composition it is helpful to understand the following theorem from Milner, Parrow and Walker, [18].

Theorem 3.5.10 *Let $P \equiv \Sigma_i \alpha_i.P_i$ and $Q \equiv \Sigma_j \beta_j.Q_j$, where α_i, β_j are finite π-calculus prefixes and no α_i (resp β_j) binds a name free in Q (resp P); then*

$$P|Q \sim_l \Sigma_i \alpha_i.(P_i|Q) + \Sigma_j \beta_j.(P|Q_j) + \Sigma_{\alpha_i * \beta_j} \begin{cases} 0 & \text{if } subj(\alpha_i) \neq subj(\beta_j) \\ \tau.R_{ij} & \text{if } subj(\alpha_i) = subj(\beta_j) \end{cases}$$

where the relation $\alpha_i * \beta_j$ (α_i complementary β_j) holds in four cases:

- *α_i is $\overline{x_i}\langle u \rangle$ and β_j is $y_j(v)$; then R_{ij} is $P_i|Q_j[v := u]$.*
- *α_i is $\overline{x_i}(u)$ and β_j is $y_j(v)$; then R_{ij} is $\nu w(P_i[u := w]||Q_j[v := w])$, where w is not free in $\nu u P_i$ or in $\nu v Q_j$.*
- *α_i is $x_i(v)$ and β_j is $\overline{y_j}\langle u \rangle$; then R_{ij} is $P_i[v := u]|Q_j$.*
- *α_i is $x_i(v)$ and β_j is $\overline{y_j}(u)$; then R_{ij} is $\nu w(P_i[v := w]||Q_j[u := w])$, where w is not free in $\nu v P_i$ or in $\nu u Q_j$.*

Now, we shall decompose $P|Q$ into finer operations based on the late-bisimulation above.

Definition 3.5.11 To properly axiomatize the parallel composition operator, we define two new operators *left merge*, denote by $P \lfloor Q$, and *synchronization*, denoted by $P||Q$.

1. $P \lfloor Q$, expresses a process which commits itself to the behavior of the prefix which guards P. For example, this signifies that $\overline{x}\langle z \rangle.P' \lfloor u(v).Q' = \overline{x}\langle z \rangle.(P'|u(v).Q')$. From the theorem above we notice that $P \lfloor Q$ represents $\Sigma_i \alpha_i.(P_i|Q)$ and $Q \lfloor P$ represents $\Sigma_j \beta_j.(P|Q_j)$.
2. $P||Q$, expresses a process which is ready for a τ transition. Therefore if P and Q are guarded by complementary prefixes $P||Q$ will be equal to the process after a τ commitment. In any other situation it shall be equivalent to the 0 process. In the above theorem we have that $P||Q = \Sigma_{\alpha_i * \beta_j} \begin{cases} 0 & \text{if } subj(\alpha_i) \neq subj(\beta_j) \\ \tau.R_{ij} & \text{if } subj(\alpha_i) = subj(\beta_j) \end{cases}$

We now have all the tools necessary to define the axioms and inference rules for the strong ground equivalence relation. Therefore SGE $\vdash P = Q$ signifies that P is equivalent to Q through use of the axioms and inference rules below. Our main result in this chapter, Proposition 3.5.13, shows these axioms characterize $\sim_l$.

Definition 3.5.12 Define *Strong Ground Equivalence (= SGE)* to be the relation generated by the following axioms and inference rules, as described in [9]:

1. The set of axioms:

   (a) *Alpha-conversion*

   A If $P \equiv_\alpha Q$, then $SGE \vdash P = Q$

   (b) *Summation*

   S1 $SGE \vdash P + 0 = P$

   S2 $SGE \vdash P + Q = Q + P$

   S3 $SGE \vdash P + (Q + R) = (P + Q) + R$

   S4 $SGE \vdash P + P = P$

   (c) *Restriction*

   R1 $SGE \vdash \nu z(P + Q) = \nu zP + \nu zQ$

   R2 if $z \notin names(\pi)$ then $SGE \vdash \nu z(\pi.P) = \pi.\nu zP$

   R3 if $z = subject(\pi)$ then $SGE \vdash \nu z(\pi.P) = 0$

   R4 if $z \neq x$ then $SGE \vdash \nu z(\overline{x}(z).P) = \overline{x}(z).P$

R5 $SGE \vdash \nu z 0 = 0$

(d) *Parallel*

PAR $SGE \vdash P|Q = P\lfloor Q + Q\lfloor P + P||Q$

(e) *Left Merge*

LM1 $SGE \vdash 0\lfloor R = 0$

LM2 $SGE \vdash (P+Q)\lfloor R = P\lfloor R + Q\lfloor R$

LM3 if $bn(\pi.0) \notin fn(Q)$ then $SGE \vdash (\pi.P)\lfloor Q = \pi.(P|Q)$

(f) *Synchronization*

SYN1 $SGE \vdash 0||P = 0$

SYN2 $SGE \vdash P||Q = Q||P$

SYN3 $SGE \vdash (P+Q)||R = P||R + Q||R$

SYN4 $SGE \vdash x(z).P||\overline{x'}\langle y\rangle.Q = \begin{cases} \tau.(P[z := y]||Q) & \text{if } x = x' \\ 0 & \text{else} \end{cases}$

SYN5 $SGE \vdash x(z).P||\overline{x'}\langle z\rangle.Q = \begin{cases} \tau.(\nu z(P|Q)) & \text{if } x = x' \\ 0 & \text{else} \end{cases}$

SYN6 $SGE \vdash \tau.P||\pi.Q = 0$

SYN7 $SGE \vdash x(z).P||x'(z).Q = 0$

SYN8 if $\pi, \pi' \in \{\overline{x}\langle y\rangle, \overline{x'}(z)\}$, then $SGE \vdash \pi.P||\pi'.Q = 0$

2. The set of inference rules:

   (a) *Equivalence*

   REF

$$SGE \vdash P = P$$

   SYMM

$$\frac{SGE \vdash P = Q}{SGE \vdash Q = P}$$

   TRANS

$$\frac{SGE \vdash P = Q \quad SGE \vdash Q = R}{SGE \vdash P = R}$$

   (b) *Congruence*

   CON1

$$\frac{SGE \vdash P = Q}{SGE \vdash \overline{x}\langle y\rangle.P = \overline{x}\langle y\rangle.Q}$$

   CON2

$$\frac{SGE \vdash P = Q}{SGE \vdash \overline{x}(z).P = \overline{x}(z).Q}$$

   CON3 Let $fn(x(z).P, x(z).Q) = \{a_1, \ldots, a_n\}$

$$\frac{\forall\, i \in n \;\; SGE \vdash P[z := a_i] = Q[z := a_i] \quad SGE \vdash P = Q}{SGE \vdash x(z).P = x(z).Q}$$

   CON4

$$\frac{SGE \vdash P = Q}{SGE \vdash \tau.P = \tau.Q}$$

   CON5

$$\frac{SGE \vdash P = Q}{SGE \vdash P + M = Q + M}$$

   CON6

$$\frac{SGE \vdash P = Q}{SGE \vdash \nu z P = \nu z Q}$$

   CON7

$$\frac{SGE \vdash P = Q}{SGE \vdash P \downarrow R = Q \downarrow R}$$

CON8

$$\frac{SGE \vdash P = Q}{SGE \vdash P||R = Q||R}$$

Hence, the above axiomatization leads to the following result, shown in [18].

Proposition 3.5.13 *Let P and Q be two finite π-calculus processes then $SGE \vdash P = Q$ if and only if $P \sim_l Q$.*

Therefore, the formal theory SGE axiomatizes late-bisimularity of processes. This will help us in the next chapter, when we try to prove that our model satisfies late-bisimulation.

Chapter 4

Presheaf Models of the Finite π-calculus

In this chapter we will discuss the properties presheaf categories must admit to give a sound interpretation of the finite π-calculus. The interpretation is based on a combination of two papers [24, 9], which discuss fully abstract models for the full π-calculus. In their work Fiore et al, [9], and Stark, [24], construct fully abstract models for the full π-calculus, using the functor category $\mathbf{Cpo}^{\mathbf{Inj}}$. Furthermore, both [24, 9], and in [11] the authors suggest the existence of a set-theoretical model for the finite fragment of the π-calculus. They also suggest that the model be constructed using the presheaf category $\mathbf{Set}^{\mathbf{Inj}}$.

Below we shall delineate some abstract properties of the functor category $\mathbf{Set}^{\mathbf{Inj}}$ which allow it to give a sound interpretation of the finite π-calculus. We will discuss the properties in **Inj** and **Set** which make them the obvious choice for such a model. Hence we shall list the abstract properties we need, then in Theorem 4.1.10 we shall prove that our concrete model $\mathbf{Set}^{\mathbf{Inj}}$ satisfies the discussed properties.

4.1 The General Category

4.1.1 Abstract properties for the model $\mathbf{Set}^{\mathbf{Inj}}$

1. **Inj** is a symmetric monoidal category, with structure $(\mathbf{Inj}, +, \emptyset)$, where $+$ represents the disjoint union. See Example 2.2.6 in Chapter 2.

2. (i) **Set** is a ccc equipped with the terminal object 1, representing an arbitrary singleton set. To differentiate between different singleton sets we shall use the notation that $1_a = \{a\}$. We consider $1=\{z\}$, unless otherwise stated.

   (ii) **Set** admits the free-semilattice monad, seen in Section 2.6.

   Remark 4.1.1 Notice that the terminal object $1 \in |\mathbf{Set}|$ is also an object of **Inj**, i.e. $1 \in |\mathbf{Inj}|$. However, 1 is not a terminal object in the category **Inj**, since for any set $s \in |\mathbf{Inj}|$ with $|s| \geq 2$ there exists no injective map from s to 1.

3. The functor category $\mathbf{Set}^{\mathbf{Inj}}$ is a dcc with symmetric monoidal structure $(\mathbf{Set}^{\mathbf{Inj}}, \otimes_{Day}, I_{Day}, \multimap_{Day})$ obtained by using the Day construction, as seen in Section 2.4.5 and a cartesian closed structure $(\mathbf{Set}^{\mathbf{Inj}}, \times, \Rightarrow, 1_{\mathbf{Set}^{\mathbf{Inj}}})$, where $\times$ is the cartesian product of **Set** taken pointwise, and $1_{\mathbf{Set}^{\mathbf{Inj}}}$ is the terminal object of $\mathbf{Set}^{\mathbf{Inj}}$ determined by the terminal object 1 of **Set**.

4. In our general presheaf model, for each object $s \in |\mathbf{Inj}|$, there exists an embedding morphism, emb_s, from the tensor product $\otimes_{Day}$ of $\mathbf{Set}^{\mathbf{Inj}}$ to the cartesian product $\times$ of $\mathbf{Set}^{\mathbf{Inj}}$. Let $F, G \in |\mathbf{Set}^{\mathbf{Inj}}|$:

   $$emb_s : (F \otimes_{Day} G)(s) \hookrightarrow (F \times G)(s)$$

   This map will induce a surjection, $surj_s$, from the function space $(F \Rightarrow G)(s)$ to the linear function space $(F \multimap G)(s)$, seen in Lemma 4.1.3 below.

5. There exists an object $N \in |\mathbf{Set}^{\mathbf{Inj}}|$, called the *object of names* which satisfies the following equations: For any $F, G \in |\mathbf{Set}^{\mathbf{Inj}}|$ and $s, s' \in |\mathbf{Inj}|$,

   (a) $N(1) \cong 1$,

(b) $N(s+s') \cong N(s) + N(s')$, where $s, s' \in |\mathbf{Inj}|$,

(c) $(N \Rightarrow F)(s) \cong F^{|N(s)|}(s) \times F(s+1)$,

(d) $(N \multimap F)(s) \cong F(s+1)$.

Informally, this functor shall take an object $s \in |\mathbf{Inj}|$ to an object of the category **Set**, which is isomorphic to some finite subset of *Act*, the set of action labels of the finite π-calculus. The functor $N : \mathbf{Inj} \to \mathbf{Set}$ represents the only exponential we shall need in our interpretation.

Remark 4.1.2 Since any finite set is isomorphic to the disjoint union of arbitrary singletons, note that if $N(1) \cong 1$ and N preserves disjoint that:

$$N(s) \cong N(1+1+\ldots+1) \cong N(1)+N(1)+\ldots+N(1) \cong 1+1+\ldots+1 \cong s \ \forall s \in |\mathbf{Inj}|$$

Therefore, we may suppose, due to the above properties that $N(s) \cong s$ for any N satisfying $(a) - (d)$.

6. There exists an object $Proc \in |\mathbf{Set}^{\mathbf{Inj}}|$, called the *object of processes*, such that it is the least fixed point of the functor:

$$\begin{array}{rccl} G: & \mathbf{Set}^{\mathbf{Inj}} & \to & \mathbf{Set}^{\mathbf{Inj}} \\ & X & \mapsto & \mathcal{P}_f^{\mathbf{Inj}}(N \times (N \Rightarrow X) + N \times N \times X + N \times (N \multimap X) + X) \end{array}$$

where the functor $\mathcal{P}_f^{\mathbf{Inj}}$ is the lifted free-semilattice monad on the category $\mathbf{Set}^{\mathbf{Inj}}$, i.e. the free-semilattice monad as stated in Definition 2.6.7 on **Set** taken pointwise. Each term in the sum respectively gives meaning to inputs, outputs, bound outputs and silent actions of processes. The use of the free-semilattice monad will enable us to interpret a subset of processes of the various types as a nondeterministic sum, i.e. when we pick an element of the subset we must disregard the remaining elements for the remainder of the process interaction.

Informally, on an object $s \in |\mathbf{Inj}|$, $Proc(s)$ will represent all the processes one can construct from the free names represented by $N(s)$. Moreover, for a map $i : s \to s'$ in **Inj**, $Proc(i) : Proc(s) \to Proc(s')$, represents a relabelling map

of processes defined on the free names from $N(s)$ to a process with free names from $N(s')$, as one can see in the equation

$$\begin{aligned} Proc(i) &= \mathcal{P}_f^{\mathbf{Inj}}(N \times (N \Rightarrow Proc) + N \times N \times Proc + N \times (N \multimap Proc) \\ &\quad + Proc)(i) \\ &= \mathcal{P}_f(N(i) \times (N \Rightarrow Proc)(i) + N(i) \times N(i) \times Proc(i) + \\ &\quad N(i) \times (N \multimap Proc)(i) + Proc(i)) \\ &\cong \mathcal{P}_f(i \times (N \Rightarrow Proc)(i) + i \times i \times Proc(i) + i \times Proc([i+1]) \\ &\quad + Proc(i)) \end{aligned}$$

We notice that every free name in these sets of processes shall be substituted by its image under the map i, hence the relabelling.

7. As seen in [9], we shall need to interpret the term constructors for our syntax of the finite π-calculus. The interpretation consists of combining the objects of names and objects of processes via natural transformations. The natural transformations are straightforward and represent the metalanguage in which we represent processes in our model.

   (a) $nil : Proc$ or equivalently $nil : 1_{\mathbf{Set}}^{\mathbf{Inj}} \stackrel{\cdot}{\to} Proc$
   This will construct the 0 process. For any object $s \in |\mathbf{Inj}|$, we have that $nil(s) = nil_s = \emptyset \in Proc(s)$

   (b) $in : N \times (N \Rightarrow Proc) \stackrel{\cdot}{\to} Proc$
   This will construct an input prefixed process. For any $s \in |\mathbf{Inj}|$, we have that $in_s : N(s) \times (N \Rightarrow Proc)(s) \to Proc(s)$ or equivalently by property 5(a) above $in_s : N(s) \times Proc^{|N(s)|}(s) \times Proc(s+1) \to Proc(s)$. Informally, the natural transformation takes as input the triple $(x, \eta : N(s) \to Proc(s), Q)$, where $x \in N(s)$ represents the subject of the input prefix, $\eta : N(s) \to Proc(s)$ represents a map which for each name in the set $N(s)$ associates a process in $Proc(s)$, and Q is a process such that $fn(Q) \subseteq N(s+1)$, (hence it may contain the local name z). Thus, it outputs a process equivalent to $x(z).P \in Proc(s)$.

(c) $out : N \times N \times Proc \stackrel{\cdot}{\to} Proc$

This will construct an output prefixed process. For any $s \in |\mathbf{Inj}|$, we have that $out_s : N(s) \times N(s) \times Proc(s) \to Proc(s)$. Hence, we can think of out_s as a natural transformation which takes as input the triple (x, y, P) where the pair of names $(x, y) \in N(s) \times N(s)$ and $P \in Proc(s)$ is a process. Thus it outputs a process equivalent to $\overline{x}\langle y \rangle.P \in Proc(s)$.

(d) $bout : N \times (N \multimap Proc) \stackrel{\cdot}{\to} Proc$

This will construct a bound output prefixed process. For any $s \in |\mathbf{Inj}|$, we have that $bout_s : N(s) \times (N \multimap Proc)(s) \to Proc(s)$ or equivalently by property 5(b), $bout_s : N(s) \times Proc(s+1) \to Proc(s)$. Hence, $bout_s$ is the natural transformation that takes as input the pair (x, Q), where $x \in N(s)$ is the subject of the bound output prefix, and a process $Q \in Proc(s+1)$ with a local name z. Thus, it outputs a process equivalent to $\overline{x}(z).Q \in Proc(s)$.

(e) $tau : Proc \stackrel{\cdot}{\to} Proc$

This will construct a silent action prefixed process. For any $s \in |\mathbf{Inj}|$, we have that $tau_s : Proc(s) \to Proc(s)$. Hence informally, we have that tau_s takes as input a process $P \in Proc(s)$ and returns a process equivalent to $\tau.P \in Proc(s)$.

(f) $res : (N \multimap Proc) \stackrel{\cdot}{\to} Proc$

This will construct a restricted process. For any $s \in |\mathbf{Inj}|$ the natural transformation $res_s : (N \multimap Proc)(s) \to Proc(s)$ or equivalently by isomorphism 5(b) seen above, $res_s : Proc(s+1) \to Proc(s)$. The natural transformation res_s takes as input the process $Q \in Proc(s+1)$, with local name z as seen in (d). Thus, it outputs a process equivalent to $\nu z P \in Proc(s)$.

(g) $sum : Proc \times Proc \stackrel{\cdot}{\to} Proc$

This will construct a non-deterministic sum process. On any object $s \in |\mathbf{Inj}|$, we get the natural transformation $sum_s : Proc(s) \times Proc(s) \to Proc(s)$. Informally, sum_s takes as input a pair of processes $P, P' \in$

$Proc(s)$ and returns their non-deterministic sum $P + P' \in Proc(s)$.

(h) $par : Proc \times Proc \stackrel{\cdot}{\to} Proc$
This will construct a parallel process. Hence for objects $s \in |\mathbf{Inj}|$, we have that $par_s : Proc(s) \times Proc(s) \to Proc(s)$. We may think of par_s as a natural transformation which takes as input a pair of processes $P, P' \in Proc(s)$ and returns their composition $P|P' \in Proc(s)$.

(i) $lm : Proc \times Proc \stackrel{\cdot}{\to} Proc$
This will construct a left merge process. For any $s \in |\mathbf{Inj}|$, we have that $lm_s : Proc(s) \times Proc(s) \to Proc(s)$. We may think of lm_s as a natural transformation which takes as input a pair of processes $P, P' \in Proc(s)$ and returns their left merge $P \lfloor P' \in Proc(s)$.

(j) $syn : Proc \times Proc \stackrel{\cdot}{\to} Proc$
This will construct a synchronization process. Hence for objects $s \in |\mathbf{Inj}|$, we have that $syn_s : Proc(s) \times Proc(s) \to Proc(s)$. We may think of syn_s as a natural transformation which inputs a pair of processes $P, P' \in Proc(s)$ and returns their synchronization $P||P' \in Proc(s)$.

8. To have our interpretation preserve the late-bisimulation relation from Section 3.5.1 on finite π-calculus processes, we need the above natural transformations to satisfy some equations. These equations will allow the interpretation of the process P and Q to be equal whenever they are late-bisimular. These equations are based on the axioms of the Strong Ground Equivalence, from Section 3.5.3. Let $s, t \in |\mathbf{Inj}|; N(s) = \{x_1, x_2, \ldots, x_n\}; z \in N(1); P, P', P'' \in Proc(s); Q, Q' \in Proc(s+1)$; and $R \in Proc((s+1)+1))$.

(a) Restriction

R1 $res_s(in_{s+1}(\alpha_i, \eta, R)) = \begin{cases} nil_s & \text{if } \alpha_i = z \\ in_s(\alpha_i, \eta_{res}, res_s(R)) & \text{else} \end{cases}$
where $\eta : N(s+1) \to Proc(s+1)$ and $\eta_{res} : N(s) \to Proc(s)$, such that $\eta_{res}(x_i) = res_s(\eta(x_i))$.

R2 $$res_s(out_{s+1}(\alpha_i, \alpha_j, Q)) = \begin{cases} nil_s & \text{if } \alpha_i = z \\ bout_s(\alpha_i, Q) & \text{if } \alpha_i \neq z \text{ and } \alpha_j = z \\ out_s(\alpha_i, \alpha_j, res_s(Q)) & \text{else} \end{cases}$$

R3 $$res_s(bout_{s+1}(\alpha_i, R)) = \begin{cases} nil_s & \text{if } \alpha_i = z \\ bout_s(\alpha_i, res_s(R)) & \text{else} \end{cases}$$

R4 $res_s(tau_{s+1}(Q)) = tau_s(res_s(Q))$

R5 $res_s(sum_{s+1}(Q, Q')) = sum_s(res_s(Q), res_s(Q'))$

R6 $res_s(nil_{s+1}) = nil_s$

(b) Summation

S1 $sum_s(P, nil_s) = P$

S2 $sum_s(P, P') = sum_s(P', P)$

S3 $sum_s(P, sum_s(P', P'')) = sum_s(sum_s(P, P'), P'')$

S4 $sum_s(P, P) = P$

(c) Parallel

P1 $par_s(P, P') = sum_s(lm_s(P, P'), lm_s(P', P), syn_s(P, P'))$

(d) Left Merge

LM1 $lm_s(nil_s, P) = nil_s$

LM2 $lm_s(sum_s(P, P'), P'') = sum_s(lm_s(P, P''), lm_s(P', P''))$

LM3 $lm_s(in_s(x_i, \eta, Q), P') = in_s(x_i, \eta_{lm}, par_{s+1}(Q, P'))$
where $\eta, \eta_{lm} : N(s) \to Proc(s)$, and $\eta_{lm}(x_i) = par_s(\eta(x_i), P')$

LM4 $lm_s(out_s(x_i, x_j, P), P') = out_s(x_i, x_j, par_s(P, P'))$

LM5 $lm_s(bout_s(x_i, Q), P') = bout_s(x_i, par_{s+1}(Q, P'))$

LM6 $lm_s(tau_s(P), P') = tau_s(par_s(P, P'))$

(e) Synchronization

SYN1 $syn_s(nil_s, P) = nil_s$

SYN2 $syn_s(P, P') = syn_s(P', P)$

SYN3 $syn_s(sum_s(P, P'), P'') = sum_s(syn_s(P, P''), syn_s(P', P''))$

SYN4 $syn_s(in_s(x_i, \eta, Q), out_s(x_j, x_k, P))$
$$= \begin{cases} nil_s & \text{if } x_i \neq x_j \\ tau_s(par_s(\eta(x_k), P)) & \text{else} \end{cases}$$
where $\eta : N(s) \to Proc(s)$.

SYN5 $syn_s(in_s(x_i, \eta, Q), bout_s(x_j, Q'))$
$$= \begin{cases} nil_s & \text{if } x_i \neq x_j \\ tau_s(res_s(par_{s+1}(Q, Q'))) & \text{else} \end{cases}$$

SYN6 $syn_s(tau_s(P), \phi) = nil_s$
where $\phi \in \{in_s(x_i, \eta, Q), out_s(x_i, x_j, P), bout_s(x_i, Q), tau_s(P)\}$,
and $\eta : N(s) \to Proc(s)$.

SYN7 $syn_s(in_s(x_i, \eta, Q), in_s(x_j, \eta', Q')) = nil_s$
where $\eta, \eta' : N(s) \to Proc(s)$.

SYN8 $syn_s(\phi, \psi) = nil_s$
where $\phi, \psi \in \{out_s(x_i, x_j, P), bout_s(x_k, Q)\}$.

Lemma 4.1.3 *For $F, G \in |\mathbf{Set}^{\mathbf{Inj}}|$ and $s \in |\mathbf{Inj}|$, the embedding map $emb_s : (F \otimes_{Day} G)(s) \to (F \times G)(s)$ induces a surjective map $surj_s : (F \Rightarrow G)(s) \to (F \multimap G)(s)$.*

Proof. Since $\mathbf{Set}^{\mathbf{Inj}}$ is a dcc, and the map $emb : H \otimes_{Day} F \stackrel{\cdot}{\to} H \times F$ induces a map $h_G(emb) : Hom_{\mathbf{Set}^{\mathbf{Inj}}}(H \times F, G) \to Hom_{\mathbf{Set}^{\mathbf{Inj}}}(H \otimes F, G)$, we have the following diagram:

$$\begin{array}{ccc} Hom_{\mathbf{Set}^{\mathbf{Inj}}}(H \otimes_{Day} F, G) & \stackrel{\lambda^*}{\cong} & Hom_{\mathbf{Set}^{\mathbf{Inj}}}(H, F \multimap G) \\ \uparrow{\scriptstyle h_G(emb)} & & \\ Hom_{\mathbf{Set}^{\mathbf{Inj}}}(H \times F, G) & \stackrel{\lambda}{\cong} & Hom_{\mathbf{Set}^{\mathbf{Inj}}}(H, F \Rightarrow G) \end{array}$$

We now use the technique from the Yoneda Lemma. Let $H = F \Rightarrow G$, this transforms the diagram to:

$$\begin{array}{ccc} Hom_{\mathbf{Set}^{\mathbf{Inj}}}((F \Rightarrow G) \otimes_{Day} F, G) & \stackrel{\lambda^*}{\cong} & Hom_{\mathbf{Set}^{\mathbf{Inj}}}(F \Rightarrow G, F \multimap G) \\ \uparrow{\scriptstyle h_G(emb)} & & \\ Hom_{\mathbf{Set}^{\mathbf{Inj}}}((F \Rightarrow G) \times F, G) & \stackrel{\lambda}{\cong} & Hom_{\mathbf{Set}^{\mathbf{Inj}}}(F \Rightarrow G, F \Rightarrow G). \end{array}$$

Hence, from the identity map $id_{F \Rightarrow G} \in Hom_{\mathbf{Set}^{\mathbf{Inj}}}(F \Rightarrow G, F \Rightarrow G)$, $\lambda^* \circ h_G(emb) \circ \lambda(id_{F \Rightarrow G}) = \lambda^* \circ h_G(emb)(eval) = \lambda^*(eval \circ emb) \in Hom_{\mathbf{Set}^{\mathbf{Inj}}}(F \Rightarrow G, F \multimap G)$, where $eval : (F \Rightarrow G) \times F \stackrel{\cdot}{\rightarrow} G$ is the evaluation map defined by the adjunction of the cartesian product. We will show that $(\lambda^*(eval \circ emb))_s$ is surjective for every $s \in |\mathbf{Inj}|$.

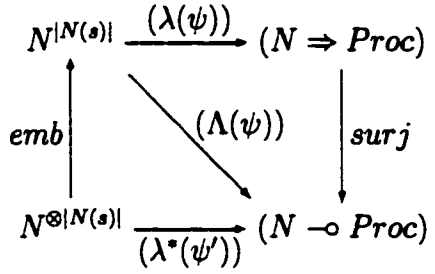
First, our map is defined to be $(((\lambda^*(eval \circ emb))_s(\theta))_{s'})(a) = \theta_{s'}(a)$, by the properties of linear currying λ^* and of the evaluation map *eval*. It is important that one convinces oneself that $(\theta, a) \in ((F \Rightarrow G) \otimes_{Day} F)(s)$, which can be easily checked. Therefore, our map is surjective since for any $\psi \in (F \multimap G)(s)$, pick $\theta \in (F \Rightarrow G)(s)$, such that $\theta_{s'}(a) = \psi_{s'}(a)$, for all $a \in F(s') \subseteq F(s + s')$. □

4.1.2 Open Interpretation

As in [9] and [24], we shall begin by giving an open interpretation of the finite π-calculus. The open interpretation will take a finite π-calculus process to a morphism in the category $\mathbf{Set}^{\mathbf{Inj}}$. More specifically, a process P such that $fn(P) \subseteq N(s)$, denoted by $N(s) \vdash P$, will be interpreted as the natural transformation $[\![N(s) \vdash P]\!] : N^{|N(s)|} \stackrel{\cdot}{\rightarrow} Proc$. Therefore, the open interpretation considers only the prefix and operational structure of the processes but does not distinguish between two processes which differ up to free names.

Lemma 4.1.4 *Let λ be the currying operation in the category $\mathbf{Set}^{\mathbf{Inj}}$, such that for a morphism $\psi : N^{|N(s)|+1} \stackrel{\cdot}{\rightarrow} Proc$ we have that $\lambda\psi : N^{|N(s)|} \stackrel{\cdot}{\rightarrow} (N \Rightarrow Proc)$ and λ^* be the linear currying operation such that for the morphism $\psi' : N^{\otimes(|N(s)|+1)} \stackrel{\cdot}{\rightarrow} Proc$ we have that $\lambda^*\psi' : N^{\otimes|N(s)|} \stackrel{\cdot}{\rightarrow} (N \multimap Proc)$. Then there exists the operator Λ such that $\Lambda = surj \circ \lambda$, where $surj_s$ is the surjection from $(N \Rightarrow Proc)(s)$ to $(N \multimap Proc)(s)$ defined in Lemma 4.1.3, such that for any $\psi : N^{|N(s)|+1} \stackrel{\cdot}{\rightarrow} Proc$ we have that $(\Lambda\psi)_s : N^{|N(s)|}(s) \rightarrow (N \multimap Proc)(s)$*

Proof. This is due to the following diagram:



□

Definition 4.1.5 (Open Interpretation) Let $N(s) \vdash P$ signify $fn(P) \subseteq N(s)$. We shall define an operator

$$[\![-]\!] : A_s^\pi \to Hom_{\mathbf{Set}^{\mathbf{Inj}}}(N^{|N(s)|}, Proc)$$

where $A_s^\pi = \{P \in Proc_f^\pi; fn(P) \subseteq s\}$. This operator takes a finite π-calculus process P such that $fn(P) \subseteq N(s)$, to a corresponding natural transformation $N^{|N(s)|} \stackrel{\cdot}{\to} Proc$. In the open interpretation we do not distinguish between the use of different free names in processes, i.e. $\overline{x_i}\langle x_j\rangle.P$ and $\overline{x_k}\langle x_l\rangle.P$ are interpreted by the same natural transformation. We define the open interpretation recursively for the finite π-calculus as follows: Let $N(s) = \{x_1, x_2, \ldots, x_n\}$ and $z \in N(1)$

1. We define the open interpretation inductively starting with a free name $x_i \in N(s)$.
$$[\![N(s) \vdash x_i]\!] : N^{|N(s)|} \stackrel{\cdot}{\to} N, \text{ is the } i^{th} \text{ projection}$$
2. $[\![N(s) \vdash 0]\!] : N^{|N(s)|} \stackrel{\cdot}{\to} Proc$ is the constant natural transformation such that for any $s' \in |\mathbf{Inj}|$, $[\![N(s) \vdash 0]\!]_{s'} = nil_s$.
3. $[\![N(s) \vdash x_i(z).P]\!] = in \circ \langle [\![N(s) \vdash x_i]\!], \lambda [\![N(s+1) \vdash P]\!] \rangle$
4. $[\![N(s) \vdash \overline{x_i}\langle x_j\rangle.P]\!] = out \circ \langle [\![N(s) \vdash x_i]\!], [\![N(s) \vdash x_j]\!], [\![N(s) \vdash P]\!] \rangle$
5. $[\![N(s) \vdash \overline{x_i}(z).P]\!] = bout \circ \langle [\![N(s) \vdash x_i]\!], \Lambda [\![N(s+1) \vdash P]\!] \rangle$
6. $[\![N(s) \vdash \tau.P]\!] = tau \circ \langle [\![N(s) \vdash P]\!] \rangle$

7. $[\![N(s) \vdash \nu z.P]\!] = res \circ \langle \Lambda [\![N(s+1) \vdash P]\!] \rangle$
8. $[\![N(s) \vdash P_1 + P_2]\!] = sum \circ \langle [\![N(s) \vdash P_1]\!], [\![N(s) \vdash P_2]\!] \rangle$
9. $[\![N(s) \vdash P_1 | P_2]\!] = par \circ \langle [\![N(s) \vdash P_1]\!], [\![N(s) \vdash P_2]\!] \rangle$
10. $[\![N(s) \vdash P_1 \downarrow P_2]\!] = lm \circ \langle [\![N(s) \vdash P_1]\!], [\![N(s) \vdash P_2]\!] \rangle$
11. $[\![N(s) \vdash P_1 || P_2]\!] = syn \circ \langle [\![N(s) \vdash P_1]\!], [\![N(s) \vdash P_2]\!] \rangle$

Lemma 4.1.6 *The following are four properties of the open interpretation. Let* $s, 1_a, 1_b, t \in |\mathbf{Inj}|$.

1. *Permutation*
   Let $N(s + 1_a + 1_b + t) \vdash P$, *then:*

$$[\![N(s + 1_a + 1_b + t) \vdash P]\!] = [\![N(s + 1_b + 1_a + t) \vdash P]\!] \circ \pi_P$$

   where $\pi_P = \langle \pi_1, \pi_2, \ldots, \pi_{|N(s)|}, \pi_{|N(s)|+2}, \pi_{|N(s)|+1}, \pi_{|N(s)|+3}, \ldots, \pi_{|N(s)|+|N(t)|+2} \rangle$

2. *Contraction*
   Let $N(s + t) \vdash P$, *and* $a' \in N(1_a)$, *then:*

$$[\![N(s + 1_a + t) \vdash P]\!] = [\![N(s + t) \vdash P]\!] \circ \pi_C$$

   where $\pi_C = \langle \pi_1, \pi_2, \ldots, \pi_{|N(s)|}, \pi_{|N(s)|+2}, \ldots, \pi_{|N(s)|+|N(t)|+1} \rangle$

3. *Substitution*
   Let $N(s + 1_a + t) \vdash P$, $a' \in N(1_a)$, *and* $x_j \in N(s + t)$, *then:*

$$[\![N(s + t) \vdash P[a' := x_j]]\!] = [\![N(s + 1_a + t) \vdash P]\!] \circ \pi_S$$

   where $\pi_S = \langle \pi_1, \pi_2, \ldots, \pi_{|N(s)|}, [\![N(s + t) \vdash x_j]\!], \pi_{|N(s)|+1}, \ldots, \pi_{|N(s)|+|N(t)|} \rangle$

4. *Alpha-Conversion*
   Let $N(s) \vdash P$ *and* $N(s) \vdash Q$. *If* $P \equiv_\alpha Q$, *then:*

$$[\![N(s) \vdash P]\!] = [\![N(s) \vdash Q]\!]$$

The proof is shown in detail in Appendix A

Corollary 4.1.7 *If* $N(s+1) \vdash Q$, *where* $z \in N(1)$, $y' \in N(1_y)$ *then*

$$\left[\!\left[N(s+1) \vdash Q \right]\!\right] = \left[\!\left[N(s+1_y) \vdash Q[z := y'] \right]\!\right]$$

Unfortunately, the open interpretation will not correspond the late-bisimulation relation, because the operational semantics considers processes with different free names semantically different. However, it will correspond to the late-equivalence relation, which forms a congruence relation over finite π-calculus processes

Example 4.1.8 From Example 3.5.6, we have seen how the processes $P = x(z).0|\overline{y}\langle u\rangle.0$ and $Q = x(z).\overline{y}\langle u\rangle.0 + \overline{y}\langle u\rangle.x(z).0$ are late-bisimular, i.e. $P \sim_l Q$. However, $P[y := x] \neq Q[y := x]$, hence $\left[\!\left[N(s) \vdash P \right]\!\right] \neq \left[\!\left[N(s) \vdash Q \right]\!\right]$.

Nevertheless,we are mainly interested in capturing late-bisimulation in our interpretation. That is why, in the next section, we shall extend our open interpretation into a closed interpretation, which shall differentiate between distinct free names and be preserved under late-bisimulation.

4.1.3 Closed Interpretation

Our open interpretation does not capture the notion of late-bisimulation in the finite π-calculus because it cannot differentiate between free names. For this reason we introduce the closed interpretation function:

$$(\!|-|\!) : A_s^\pi \to Proc(s)$$

where $A_s^\pi = \{P \in Proc_f^\pi; fn(P) \subseteq s\}$. This interpretation sends a finite π-calculus process with free names in $N(s)$ to an element of $Proc(s)$. Therefore, we shall interpret free names by $(\!|N(s) \vdash x_i|\!) = x_i \in N(s)$ and processes by $(\!|N(s) \vdash P|\!) \in Proc(s)$. We shall define the closed interpretation by recursion:
Let $N(s) = \{x_1, x_2, \ldots, x_n\}$, $z \in N(1)$, and $P, P_i \in Proc(s)$,

1. $(\!|N(s) \vdash x_i|\!) = x_i$

2. $(\!|N(s) \vdash 0|\!) = nil_s$

3. $(\!|N(s) \vdash x_i(z).P|\!) = in_s \circ \langle (\!|N(s) \vdash x_i|\!), \langle (\!|N(s) \vdash P[z := x_i]|\!) \rangle_{i=1}^{n}, (\!|N(s+1) \vdash P|\!) \rangle$

4. $(\!|N(s) \vdash \overline{x_i}\langle x_j \rangle.P|\!) = out_s \circ \langle (\!|N(s) \vdash x_i|\!), (\!|N(s) \vdash x_j|\!), (\!|N(s) \vdash P|\!) \rangle$

5. $(\!|N(s) \vdash \overline{x_i}(z).P|\!) = bout_s \circ \langle (\!|N(s) \vdash x_i|\!), (\!|N(s+1) \vdash P|\!) \rangle$

6. $(\!|N(s) \vdash \tau.P|\!) = tau_s \circ \langle (\!|N(s) \vdash P|\!) \rangle$

7. $(\!|N(s) \vdash \nu z.P|\!) = res_s \circ \langle (\!|N(s+1) \vdash P|\!) \rangle$

8. $(\!|N(s) \vdash P_1 + P_2|\!) = sum_s \circ \langle (\!|N(s) \vdash P_1|\!), (\!|N(s) \vdash P_2|\!) \rangle$

9. $(\!|N(s) \vdash P_1 | P_2|\!) = par_s \circ \langle (\!|N(s) \vdash P_1|\!), (\!|N(s) \vdash P_2|\!) \rangle$

10. $(\!|N(s) \vdash P_1 \lfloor P_2|\!) = lm_s \circ \langle (\!|N(s) \vdash P_1|\!), (\!|N(s) \vdash P_2|\!) \rangle$

11. $(\!|N(s) \vdash P_1 || P_2|\!) = syn_s \circ \langle (\!|N(s) \vdash P_1|\!), (\!|N(s) \vdash P_2|\!) \rangle$

We notice that there is a similarity in the definitions of the open and closed interpretations. In fact, take the process $P = \overline{x_i}\langle x_j \rangle.0$ such that $fn(P) \subseteq N(s) = \{x_1, x_2, \ldots, x_n\}$. Since the open interpretation of the process P is a natural transformation we can observe it at the object $s \in |\mathbf{Inj}|$, i.e. $\llbracket N(s) \vdash P \rrbracket_s : N^n(s) \rightarrow Proc(s)$. Finally, since $\vec{x} = (x_1, x_2, \ldots, x_n) \in N^n(s)$, we have that $\llbracket N(s) \vdash P \rrbracket_s(x_1, x_2, \ldots, x_n) \in Proc(s)$. Therefore

$$\begin{aligned} \llbracket N(s) \vdash P \rrbracket_s \vec{x} &= out_s(\llbracket N(s) \vdash x_i \rrbracket_s \vec{x}, \llbracket N(s) \vdash x_j \rrbracket_s \vec{x}, \llbracket N(s) \vdash 0 \rrbracket_s \vec{x}) \\ &= out_s(x_i, x_j, nil_s) \end{aligned}$$

Moreover, the closed interpretation of P is :

$$\begin{aligned} (\!|N(s) \vdash P|\!) &= out_s((\!|N(s) \vdash x_i|\!), (\!|N(s) \vdash x_j|\!), (\!|N(s) \vdash 0|\!)) \\ &= out_s(x_i, x_j, nil_s) \end{aligned}$$

There is in fact a relation between the open interpretation and the closed interpretation of our finite π-calculus processes. It is described in the following lemma.

Lemma 4.1.9 *Let $N(s) = \{x_1, x_2, \ldots, x_n\}$, then for any finite π-calculus process P such that $N(s) \vdash P$,*

$$(\!|N(s) \vdash P|\!) = [\![N(s) \vdash P]\!]_s(x_1, x_2, \ldots, x_n)$$

Proof ([9]). There are two main cases we need to check: Let $s \in |\mathbf{Inj}|, \vec{x} = (x_1, \ldots, x_n), N(s) = \{x_1, x_2, \ldots, x_n\}$, and $z \in N(1)$

1. We must show that

$$(\lambda[\![N(s+1) \vdash P]\!]_s\vec{x} = (\langle (\!|N(s) \vdash P[z := x_i]|\!)\rangle_{i=1}^n, (\!|N(s+1) \vdash P|\!))$$

Given our isomorphism $\alpha : (N \Rightarrow Proc)(s) \to Proc^n(s) \times Proc(s+1)$, and since $(\lambda[\![N(s+1) \vdash P]\!]_s\vec{x} \in (N \Rightarrow Proc)(s)$, $\alpha(\lambda[\![N(s+1) \vdash P]\!]_s\vec{x}) \in Proc^n(s) \times Proc(s+1)$. By Proposition 4.2.2 the map α defined by $\alpha(\theta) = \langle \theta_s(id_s, -), \theta_{s+1}(e, z)\rangle$, where $e : s \hookrightarrow s+1$ is an embedding and $z \in N(1)$, is an adequate isomorphism from $(N \Rightarrow Proc)(s)$ to $Proc^n(s) \times Proc(s+1)$. Therefore, we can compute the following:

$$\begin{aligned}
& (\lambda[\![N(s+1) \vdash P]\!]_s\vec{x} \\
= \; & \langle((\lambda[\![N(s+1) \vdash P]\!]_s(\vec{x}))_s(id_s, -), ((\lambda[\![N(s+1) \vdash P]\!]_s(\vec{x}))_{s+1}(e, z)\rangle \\
= \; & \langle((\lambda[\![N(s+1) \vdash P]\!]_s(\vec{x}))_s(id_s, -), [\![N(s+1) \vdash P]\!]_{s+1}(N^n(e)(\vec{x}), z)\rangle \\
= \; & \langle((\lambda[\![N(s+1) \vdash P]\!]_s(\vec{x}))_s(id_s, -), [\![N(s+1) \vdash P]\!]_{s+1}(\vec{x}, z)\rangle \\
= \; & \langle((\lambda[\![N(s+1) \vdash P]\!]_s(\vec{x}))_s(id_s, -), (\!|N(s+1) \vdash P|\!)\rangle
\end{aligned}$$

Moreover, since for $1 \leq i \leq n$, Lemma 4.1.6 admits that:

$$\begin{aligned}
(\!|N(s) \vdash P[z := x_i]|\!) &= [\![N(s) \vdash P[z := x_i]]\!]_s(\vec{x}) \\
&= [\![N(s+1) \vdash P]\!]_s(\vec{x}, [\![N(s) \vdash x_i]\!]_s\vec{x}) \\
&= [\![N(s+1) \vdash P]\!]_s(\vec{x}, x_i)
\end{aligned}$$

it follows that

$$\begin{aligned}
((\lambda[\![N(s+1) \vdash P]\!]_s(\vec{x}))_s(id_s, x_i) &= [\![N(s+1) \vdash P]\!]_s(N^n(id_s)(\vec{x}), x_i) \\
&= [\![N(s+1) \vdash P]\!]_s(\vec{x}, x_i) \\
&= (\!|N(s) \vdash P[z := x_i]|\!)
\end{aligned}$$

2. We must show that

$$\Lambda'(\llbracket N(s+1) \vdash P \rrbracket) = (\!| N(s+1) \vdash P |\!)$$

However, we have previously shown in Lemma 4.1.4, that $\Lambda' = surj \circ \lambda$. Hence

$$\begin{aligned}(\Lambda' \llbracket N(s+1) \vdash P \rrbracket)_s \vec{x} &= (surj \circ \lambda \llbracket N(s+1) \vdash P \rrbracket)_s \vec{x} \\ &= surj_s \circ \langle (\!| N(s) \vdash P[z := x_i] |\!) \rangle_{x_i \in s}, (\!| N(s+1) \vdash P |\!) \rangle \\ &= (\!| N(s+1) \vdash P |\!)\end{aligned}$$

□

4.1.4 Soundness

In this section we shall demonstrate that the closed interpretation of our finite π-calculus processes will preserve the late-bisimulation relation. In other words, if two processes P and Q are late-bisimular, we shall show that $(\!| N(s) \vdash P |\!) = (\!| N(s) \vdash Q |\!)$.

Theorem 4.1.10 *Let P and Q be two finite π-calculus processes such that $N(s) \vdash P$ and $N(s) \vdash Q$, where $N(s) = \{x_1, x_2, \ldots, x_n\}$. If $SGE \vdash P = Q$, that is the processes are late-bisimular, then $(\!| N(s) \vdash P |\!) = (\!| N(s) \vdash Q |\!)$.*

The details of the proof are shown in Appendix B

4.2 A Concrete Model for the Finite π-calculus

From the properties defined in the previous section we can construct a concrete presheaf model for the finite π-calculus using the presheaf category $\mathbf{Set}^{\mathbf{Inj}}$. We now analyze in detail the previously-discussed categorical structure of $\mathbf{Set}^{\mathbf{Inj}}$ and use it to assign concrete mathematical operations to our defined natural transformations. We list the following facts using the same numbering as they appeared when stating the general list of abstract properties in Section 4.1.1.

1. and **2.** Are exactly as they were stated in Section 4.1.1.

3. $\mathbf{Set}^{\mathbf{Inj}}$ is a dcc.

(i) $\mathbf{Set}^{\mathbf{Inj}}$ is a cartesian closed category:

(a) $\mathbf{Set}^{\mathbf{Inj}}$ has products: for $F, G \in |\mathbf{Set}^{\mathbf{Inj}}|$, and $s \in |\mathbf{Inj}|$:

$$(F \times G)(s) = F(s) \times G(s)$$

where $\times$ is the product in the category **Set**.

(b) $\mathbf{Set}^{\mathbf{Inj}}$ has a corresponding function space, as previously defined, for presheaf categories. Let $F, G \in |\mathbf{Set}^{\mathbf{Inj}}|$ and $s \in |\mathbf{Inj}|$:

$$(F \Rightarrow G)(s) = Hom_{\mathbf{Set}^{\mathbf{Inj}}}(Hom_{\mathbf{Inj}}(s, -) \times F, G)$$

(ii) $\mathbf{Set}^{\mathbf{Inj}}$ is a symmetric monoidal closed category.

(a) The tensor product follows from the Day construction in Section 2.4.5. We show below it is given by the following construction of Stark adapted from [24]. Let $F, G \in |\mathbf{Set}^{\mathbf{Inj}}|$ and $s \in |\mathbf{Inj}|$:

$$\begin{aligned}(F \otimes_S G)(s) \;=\; & \{(a,b) \in (F \times G)(s); \exists f : i \rightarrow s, g : j \rightarrow s, \\ & a' \in F(i), \text{ and } b' \in G(j) \text{ such that } a = Ff(a'), \\ & b = Gg(b') \text{ and } Im(f) \cap Im(g) = \emptyset\}\end{aligned}$$

(b) The tensor product is symmetric because **Inj** is a symmetric monoidal category.

(c) The linear space corresponding to the above tensor product is defined as follows. Let $F, G \in |\mathbf{Set}^{\mathbf{Inj}}|$ and $s \in |\mathbf{Inj}|$:

$$(F \multimap G)(s) = Hom_{\mathbf{Set}^{\mathbf{Inj}}}(F, G(s + -))$$

Remark 4.2.1 Given that $A \multimap -$ is also the left adjoint for Stark's more accessible definition of the tensor product on $\mathbf{Set}^{\mathbf{Inj}}$, given by: let $F, G \in |\mathbf{Set}^{\mathbf{Inj}}|$ and $s \in |\mathbf{Inj}|$:

$$\begin{aligned}(F \otimes_{Stark} G)(s) \;=\; & \{(a,b) \in (F \times G)(s); \exists f : i \rightarrow s, g : j \rightarrow s, \\ & a' \in F(i), \text{ and } b' \in G(j) \text{ such that } a = Ff(a'), \\ & b = Gg(b') \text{ and } Im(f) \cap Im(g) = \emptyset\},\end{aligned}$$

we have that $\otimes_{Stark}$ and $\otimes_{Day}$ describe the same tensor product up to isomorphism on $\mathbf{Set}^{\mathbf{Inj}}$.

4. By the above definitions of the product and tensor we have the following facts, for $F, G \in |\mathbf{Set}^{\mathbf{Inj}}$, and $s \in |\mathbf{Inj}|$:
An embedding,

$$(F \otimes G)(s) \hookrightarrow (F \times G)(s),$$

which also forces a surjection from Lemma 4.1.3,

$$(F \Rightarrow G)(s) \twoheadrightarrow (F \multimap G)(s).$$

5. We shall define the object of names $N : \mathbf{Inj} \to \mathbf{Set}$ to be the element of $\mathbf{Set}^{\mathbf{Inj}}$ which embeds **Inj** into **Set**, i.e. for $s \in |\mathbf{Inj}|$, $N(s) = s$ and for $i : s \to s'$ in **Inj**, $N(i) = i$. Now we make sure that our functor satisfies the desired equations.

Theorem 4.2.2 *For $s \in |\mathbf{Inj}|$ we have:*

(a) $(N \Rightarrow F)(s) \cong (F^{|s|}(s)) \times F(s+1)$

(b) $(N \multimap F)(s) \cong (F(s+1))$

Proof.
(a) First, notice that $F^{|s|}(s)$ represents all the possible outputs of a function of the form $\eta : s \to F(s)$. Therefore we can represent the $|s|$-tuple $F^{|s|}(s)$ by a function η.
(i) Define a map $f : F^{|s|}(s) \times F(s+1) \to (N \Rightarrow F)(s)$ by:

$$\begin{array}{rccc} f: & F^{|s|}(s) \times F(s+1) & \to & (N \Rightarrow F)(s) \\ & (\eta : s \to F(s), \mu) & \mapsto & \theta^{(\eta,\mu)} \end{array}$$

such that

$$(\theta^{(\eta,\mu)})_{s'}(i,y) = \begin{cases} F(i)(\eta(x)) & \text{if } y = i(x), x \in s \\ F([i,y])(\mu) & \text{if } y \notin Im(i) \end{cases}$$

where $i : s \to s'$, and $[i,y] : s+1 \to s'+1$.
(ii) Define a map $g : (N \Rightarrow F)(s) \to F^{|s|}(s) \times F(s+1)$ such that:

$$\begin{array}{rccc} g: & (N \Rightarrow F)(s) & \to & F^{|s|}(s) \times F(s+1) \\ & \theta & \mapsto & ((\theta_s(id_s, -), \theta_{s+1}(e,z)) \end{array}$$

where $e : s \hookrightarrow (s+1)$, and $z \in ((s+1) - s)$. Notice $\theta_s(id_s, -) : s \to F(s)$, is the function which we have identified with $F^{|s|}(s)$.

The two maps f and g are inverse to one another:

(i) We wish to show that:

$$((f \circ g)(\theta))_{s'}(i, y) = \theta_{s'}(i, y)$$

There are two cases to check:

(1) Case 1: Suppose that $y = i(x)$, for some $x \in s$.

$$\begin{aligned} ((f \circ g)(\theta))_{s'}(i, y) &= (f(\theta_s(id_s, -), \theta_{s+1}(e, z))_{s'}(i, y) \\ &= (\theta^{(\theta_s(id_s,-),\theta_{s+1}(e,z))})_{s'}(i, y) \\ &= F(i)(\theta_s(id_s, x)) \\ &= \theta_{s'}(i, y) \end{aligned}$$

(2) Case 2: Suppose that $y \notin Im(i)$.

$$\begin{aligned} ((f \circ g)(\theta))_{s'}(i, y) &= (f(\theta_s(id_s, -), \theta_{s+1}(e, z))_{s'}(i, y) \\ &= (\theta^{(\theta_s(id_s,-),\theta_{s+1}(e,z))})_{s'}(i, y) \\ &= F([i, y])(\theta_{s+1}(e, z)) \\ &= \theta_{s'}(i, y) \end{aligned}$$

Therefore, both cases allow us to conclude that $f \circ g = id_{(N \Rightarrow F)(s)}$.

(ii) We wish to show that:

$$(g \circ f)(\eta, \mu) = (\eta, \mu)$$

$$\begin{aligned} (g \circ f)(\eta, \mu) &= g(\theta^{(\eta,\mu)}) \\ &= (\theta_s^{(\eta,\mu)}(id_s, -), \theta_{s+1}^{(\eta,\mu)}(e, z)) \end{aligned}$$

where the following equalities hold:

(1) $\theta_s^{(\eta,\mu)}(id_s, -)(x) = \theta_s^{(\eta,\mu)}(id_s, x) = F(id_s)(\eta(x)) = \eta(x)$, and

(2) $\theta_{s+1}^{(\eta,\mu)}(e, z) = F([e, z])(\mu) = \mu$.

Therefore we conclude that $g \circ f = id_{F^{|s|}(s) \times F(s+1)}$. Finally, we conclude, by the above, that we have an isomorphism.

(*b*) For $\theta \in (N \multimap F)(s) = Hom(N, F(s+-))$; we have the following square commutes:

$$\begin{array}{ccc} s' & & N(s') \xrightarrow{\theta_{s'}} F(s+s') \\ \Big\downarrow i & & \Big\downarrow Ni \qquad\qquad \Big\downarrow F(s+i) \\ s'' & & N(s'') \xrightarrow[\theta_{s''}]{} F(s+s'') \end{array}$$

Hence this is true for the following; Let $s' = \{x_1, x_2, \ldots, x_n\}$:

$$\begin{array}{ccc} 1 & & N(1) \xrightarrow{\theta_1} F(s+1) \\ \Big\downarrow i_j & & \Big\downarrow N(i_j) \qquad\qquad \Big\downarrow F(s+i_j) \\ s' & & N(s') \xrightarrow[\theta_{s'}]{} F(s+s') \end{array}$$

where we have:

$$\theta_{s'} \circ N(i)(z) = F(s+i) \circ \theta_1(z)$$

$$\theta_{s'}(x_j) = F(s+i_j)\theta_1(z)$$

Hence given $\theta \in (N \multimap F)(s)$ define $f : (N \multimap F)(s) \to F(s+1)$ by:

$$\begin{array}{rccc} f: & (N \multimap F)(s) & \to & F(s+1) \\ & \theta & \mapsto & \theta_1(z) \end{array}$$

Moreover, we also define a map:

$$\begin{array}{rccc} g: & F(s+1) & \to & (N \multimap F)(s) \\ & \mu & \mapsto & \theta^{\mu} \end{array}$$

where $\theta^{\mu} : N \stackrel{\bullet}{\to} F(s+-)$ is a natural transformation defined by:

$$\begin{array}{rccc} (\theta^{\mu})_{s'}: & N(s') & \to & F(s+s') \\ (\theta^{\mu})_{s'}: & s' & \to & F(s+s') \\ & x_j & \mapsto & F(s+i_j)\mu \end{array}$$

We verify that θ^μ is a natural transformation. Hence we must show that

$$(\theta^\mu)_{s''} \circ Ni = F(s+i) \circ (\theta^\mu)_{s'}$$

LHS

$$\begin{aligned}(\theta^\mu)_{s''} \circ Ni(x_j) &= (\theta^\mu)_{s''}(i(x_j)) \\ &= (\theta^\mu)_{s''}(y_k) \\ &= F(s+i'_k)\mu\end{aligned}$$

RHS

$$\begin{aligned}F(s+i) \circ (\theta^\mu)_{s'}(x_j) &= F(s+i) \circ F(s+i_j)\mu \\ &= F((s+i) \circ (s+i_j))\mu \\ &= F(s+i \circ i_j)\mu \\ &= F(s+i'_k)\mu\end{aligned}$$

since

$$\begin{array}{rcccrcl} i'_k: & 1 & \to & s'' & i \circ i_j: & 1 \to & s'' \\ & z & \mapsto & y_k & i \circ i_j(z) & = i(x_j) & = y_k \end{array}$$

So, since RHS = LHS, we indeed have a natural transformation.
Finally we verify that our two maps are inverses. We need that:

$$f \circ g = id_{F(s+1)} \text{ and } g \circ f = id_{(N \multimap F)(s)}$$

Let $\mu \in F(s+1)$

$$\begin{aligned}(f \circ g)(\mu) &= f(g(\mu)) \\ &= f(\theta^\mu) \\ &= (\theta^\mu)_1(z) \\ &= F(s+id_z)\mu \\ &= (id_{F(s+1)})\mu \\ &= \mu\end{aligned}$$

Let $\psi \in (N \multimap F)(s)$

$$\begin{aligned}(g \circ f)(\psi) &= g(f(\psi)) \\ &= g(\psi_1(z)) \\ &= \theta^{\psi_1(z)}\end{aligned}$$

We want that $\theta^{\psi_1(z)}$ should be equal to ψ. Therefore the following should hold:

$$(\theta^{\psi_1(z)})s'(x_j) = (\psi_{s'})(x_j)$$

LHS

$$(\theta^{\psi_1(z)})s'(x_j) = F(s+i_j)\psi_1(z)$$

RHS

$$\begin{aligned}(\psi_{s'})(x_j) &= (\psi_{s'})(Ni_j)(z)\\ &= F(s+i_j)\psi_1(z)\end{aligned}$$

This show the equality $\theta^{\psi_1(z)} = \psi$.
Therefore, we have the required isomorphism:

$$(N \multimap F)(s) \cong F(s+1)$$

□

6. $\mathbf{Set}^{\mathbf{Inj}}$ admits a lifted free-semilattice monad, namely the finite powerset monad $\mathcal{P}_f^{\mathbf{Inj}}$ taken pointwise.
Let $G : \mathbf{Set}^{\mathbf{Inj}} \to \mathbf{Set}^{\mathbf{Inj}}$, be the functor which is defined as follows, for $X \in \mathbf{Set}^{\mathbf{Inj}}$:

$$G(X) = \mathcal{P}_f^{\mathbf{Inj}}(N \times (N \Rightarrow X) + N \times N \times X + N \times (N \multimap X) + X)$$

Proposition 4.2.3 *Let Proc be the colimit of the sequence of objects* $0 \to G0 \to G^20 \to \ldots$, *where 0 is the initial object in* $\mathbf{Set}^{\mathbf{Inj}}$, *then Proc is the least fixed point for the endofunctor G.*

Proof. It will be enough to show that G preserves directed union and apply Theorem 2.7.6. Let $\cup_i A_i$ be the colimit of the directed sequence $A_1 \to A_2 \to A_3 \to \ldots$ where $A_i \in \mathbf{Set}^{\mathbf{Inj}}$ and $A_i \to A_{i+1}$ is a monomorphism. We will show that for any $s \in \mathbf{Inj}, G(\cup_i A_i)(s) = \cup_i(G(A_i)(s))$.

Suppose $X \in G(\cup_i A_i)(s)$, then X is a finite subset of

$$s \times (\cup_i A_i(s))^{|s|} \times \cup_i A_i(s+1) + s \times s \times \cup_i A_i(s) + s \times \cup_i A_i(s+1) + \cup_i A_i(s).$$

Therefore an element $x_u \in X$ is of one of the following forms

- $x_u = (x, P_1, \ldots, P_{|s|}, Q)$, where $x \in s$, $P_j \in \cup_i A_i(s)$ and $Q \in \cup_i A_i(s+1)$,
- $x_u = (x, y, P)$, where $x, y \in s$ and $P \in \cup_i A_i(s)$,
- $x_u = (x, Q)$, where $x \in s$ and $Q \in \cup_i A_i(s+1)$,
- $x_u = (P)$, where $P \in \cup_i A_i(s)$.

We will show that $X \in GA_k(s)$ for $k = max_u k_u$, where the k_u are determined by each element x_u as follows

- If $x_u = (x, P_1, \ldots, P_{|s|}, Q)$, for each $P_j \in \cup_i A_i(s)$ and $Q \in \cup_i A_i(s+1)$, there exists a least m_j such that $P_j \in A_{m_j}(s)$ and a least n such that $Q \in A_n(s+1)$. Hence, let $k_u = max(m_1, \ldots, m_{|s|}, n)$.
- If $x_u = (x, y, P)$, for $P \in \cup_i A_i(s)$, there exists a least m such that $P \in A_m(s)$. Hence, let $k_u = m$.
- If $x_u = (x, Q)$, for $Q \in \cup_i A_i(s+1)$, there exists a least m such that $Q \in A_m(s+1)$. Hence, let $k_u = m$.
- If $x_u = (P)$, for $P \in \cup_i A_i(s)$, there exists a least m such that $P \in A_m(s)$. Hence, let $k_u = m$.

Therefore, we have that $X \in GA_k(s)$ which implies $X \in \cup_i GA_i(s)$.

Suppose $X \in \cup_i GA_i(s)$, then there exists a least k such that $X \in GA_k(s)$. Since G preserves inclusions and that $A_i \subseteq A_{i+1}$ for all i, therefore $GA_i \subseteq GA_{i+1}$. Hence we consider X as a finite subset of

$$s \times A_k^{|s|}(s) \times A_K(s+1) + s \times s \times A_k(s) + s \times A_k(s+1) + A_k(s).$$

However, this makes X a finite subset of

$$s \times (\cup_i A_i(s))^{|s|} \times \cup_i A_i(s+1) + s \times s \times \cup_i A_i(s) + s \times \cup_i A_i(s+1) + \cup_i A_i(s)$$

since for any $s \in \mathbf{Inj}$, $A_k(s) \subseteq \cup_i A_i(s)$. □

Remark 4.2.4 Consider the slightly modified endofunctor $G' : \mathbf{Set}^{\mathbf{Inj}} \to \mathbf{Set}^{\mathbf{Inj}}$ such that

$$G'(X) = \mathcal{P}_f^{\mathbf{Inj}}(N \times N \times X + N \times (N \multimap X) + X).$$

Which will be used to construct an object of broadcasting processes $BProc$, i.e. processes which do not possess any input prefix. Define $BProc$ to be the least fixed point of G', found by the same method we found $Proc$. What is interesting to remark is that $BProc$ is isomorphic to the functor $F : \mathbf{Inj} \to \mathbf{Set}$ such that $F(s) = \{P \in BProc_f^{\pi}; fn(P) \subseteq s\}$, where $BProc_f^{\pi}$ represents the set of all broadcasting processes. Therefore, we have that $BProc(s)$ contains exactly the processes needed to model a broadcasting variant of the finite π-calculus.

In the case of $Proc$ we do not obtain such a strong result. However, we can claim the following:

Let $T : \mathbf{Inj} \to \mathbf{Set}$ be the functor such that $T(s) = \{P \in PRoc_f^{\pi}; fn(P) \subseteq s\}$, there is a canonical embedding α such that $\alpha : T \to Proc$. This signifies that our object of processes is larger than the set of all finite π-calculus processes. The isomorphism fails because the type $N \times (N \Rightarrow Proc)$ of G contains more elements than is needed to describe input processes.

7. $\mathbf{Set}^{\mathbf{Inj}}$ admits the necessary natural transformations to model the finite π-calculus.

1. $nil : Proc$ or equivalently $nil : 1 \stackrel{\cdot}{\to} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$\begin{array}{rccc} nil_s : & 1(s) & \to & Proc(s) \\ & \{z\} & \mapsto & \emptyset \end{array}$$

   Hence $nil_s = \emptyset$

2. $out : N \times N \times Proc \stackrel{\cdot}{\to} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$\begin{array}{rccc} out_s : & (N \times N \times Proc)(s) & \to & Proc(s) \\ : & N(s) \times N(s) \times Proc(s) & \to & Proc(s) \\ : & s \times s \times Proc(s) & \to & Proc(s) \\ & (x_i, x_j, P) & \mapsto & \{(x_i, x_j, P)\} \end{array}$$

   Hence $out_s =$ the singleton map.

3. $in : N \times (N \Rightarrow Proc) \stackrel{\cdot}{\rightarrow} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$\begin{array}{rrcl} in_s : & (N \times (N \Rightarrow Proc))(s) & \rightarrow & Proc(s) \\ : & N(s) \times (N \Rightarrow Proc)(s) & \rightarrow & Proc(s) \\ : & s \times (Proc^{|s|}(s) \times Proc(s+1)) & \rightarrow & Proc(s) \\ & (x_i, (\eta : s \rightarrow Proc(s), Q)) & \mapsto & \{(x_i, (\eta : s \rightarrow Proc(s), Q))\} \end{array}$$

Hence in_s = the singleton map.

4. $bout : N \times (N \multimap Proc) \stackrel{\cdot}{\rightarrow} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$\begin{array}{rrcl} bout_s : & (N \times (N \multimap Proc))(s) & \rightarrow & Proc(s) \\ : & N(s) \times (N \multimap Proc)(s) & \rightarrow & Proc(s) \\ : & s \times (Proc(s+1)) & \rightarrow & Proc(s) \\ & (x_i, Q) & \mapsto & \{(x_i, Q)\} \end{array}$$

Hence $bout_s$ = the singleton map.

5. $tau : Proc \stackrel{\cdot}{\rightarrow} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$\begin{array}{rrcl} tau_s : & Proc(s) & \rightarrow & Proc(s) \\ & P & \mapsto & \{(P)\} \end{array}$$

Hence tau_s = the singleton map.

6. $res : N \multimap Proc \stackrel{\cdot}{\rightarrow} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$\begin{array}{rrcl} res_s : & (N \multimap Proc)(s) & \rightarrow & Proc(s) \\ : & Proc(s+1) & \rightarrow & Proc(s) \end{array}$$

We define the restriction function $res_s(Q)$ by cases as follows:
Let $s = \{x_1, x_2, x_3, \ldots, x_n\}$ and $z \in (s+1) - s$ and $Q \in Proc(s+1)$.

(a) If $Q = \emptyset$, then Q would be representing the zero process. (i.e. $Q = nil_{s+1}$). Then

$$\begin{array}{rcl} res_s(Q) & = & res_s(nil_{s+1}) \\ & = & nil_s \end{array}$$

(b) If Q represents an output process, (i.e. $Q = \{(x_i, x_j, Q')\}$ $= out_{s+1}((x_i, x_j, Q')))$. There are three cases we must check:

i. Suppose $x_i = z$,

$$\begin{aligned} res_s(Q) &= res_s(out_{s+1}((z, x_j, Q'))) \\ &= nil_s \end{aligned}$$

ii. Suppose $x_i \neq z$, and $x_j = z$,

$$\begin{aligned} res_s(Q) &= res_s(out_{s+1}((x_i, z, Q'))) \\ &= bout_s((x_i, res_s(Q')) \end{aligned}$$

iii. Suppose $x_i, x_j \neq z$,

$$\begin{aligned} res_s(Q) &= res_s(out_{s+1}((x_i, x_j, Q'))) \\ &= out_s((x_i, x_j, res_s(Q'))) \end{aligned}$$

(c) If Q represents an input process, (i.e. $Q = \{(x_i, (\eta : s+1 \to Proc(s+1), Q'))\} = in_{s+1}((x_i, (\eta : s+1 \to Proc(s+1), Q'))))$, then there are two cases to check:

i. Suppose $x_i = z$,

$$\begin{aligned} res_s(Q) &= res_s(in_{s+1}((z, (\eta : s \to Proc(s+1), Q')) \\ &= nil_s \end{aligned}$$

ii. Suppose $x_i \neq z$,

$$\begin{aligned} res_s(Q) &= res_s(in_{s+1}((x_i, (\eta : s \to Proc(s+1), Q')) \\ &= in_s((x_i, (res_s \circ \eta : s \to Proc(s), res_s(Q')))) \end{aligned}$$

(d) If Q represents a bound output, (i.e. $Q = \{(x_i, Q')\} = bout_{s+1}((x_i, Q')))$ then there are two cases to check:

i. Suppose $x_i = z$,

$$\begin{aligned} res_s(Q) &= res_s(bout_{s+1}((z, Q')) \\ &= nil_s \end{aligned}$$

ii. Suppose $x_i \neq z$,

$$\begin{aligned} res_s(Q) &= res_s(bout_{s+1}((x_i, Q')) \\ &= bout_s((x_i, res_s(Q')) \end{aligned}$$

(e) If Q represents a silent action process, (i.e. $Q = \{(Q')\} = tau_{s+1}(Q')$) then:

$$\begin{aligned} res_s(Q) &= res_s(tau_{s+1}(Q')) \\ &= \{(res_s(Q'))\} \end{aligned}$$

7. $sum : Proc \times Proc \xrightarrow{\cdot} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$sum_s : \ Proc(s) \times Proc(s) \ \rightarrow \ Proc(s)$$

We define the summation function $sum_s(P_1, P_2)$ as follows:

$$sum_s(P_1, P_2) \ = \ \mu(P_1, P_2)$$

where μ is the big union map defined in Example 2.5.2.

8. $par : Proc \times Proc \xrightarrow{\cdot} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$par_s : \ Proc(s) \times Proc(s) \ \rightarrow \ Proc(s)$$

We define the composition function $par_s(P_1, P_2)$ as follows:

$$par_s(P_1, P_2) \ = \ sum_s(lm_s(P_1, P_2), lm_s(P_2, P_1), syn(P_1, P_2))$$

9. $lm_s : Proc \times Proc \xrightarrow{\cdot} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$lm_s : \ Proc(s) \times Proc(s) \ \rightarrow \ Proc(s)$$

We define the left merge function $lm_s(P_1, P_2)$ by cases:

(a) Suppose $P_1 = nil_s$, then

$$\begin{aligned} lm_s(P_1, P_2) &= lm_s(nil_s, P_2) \\ &= nil_s \end{aligned}$$

(b) Suppose $P_1 = in_s(x_i, \eta : s \to Proc(s), Q)$, then

$$\begin{aligned} lm_s(P_1, P_2) &= lm_s(in_s(x_i, \eta, Q), P_2) \\ &= in_s(x_i, \eta' : s \to Proc(s), par_{s+1}(Q, P_2)) \end{aligned}$$

where $\eta'(x_i) = lm_s(\eta(x_i), Q)$

(c) Suppose $P_1 = out_s(x_i, x_j, P)$, then

$$\begin{aligned} lm_s(P_1, P_2) &= lm_s(out_s(x_i, x_j, P), P_2) \\ &= out_s(x_i, x_j, par_s(P, P_2)) \end{aligned}$$

(d) Suppose $P_1 = bout_s(x_i, Q)$, then

$$\begin{aligned} lm_s(P_1, P_2) &= lm_s(bout_s(x_i, Q), P_2) \\ &= bout_s(x_i, par_{s+1}(Q, P_2)) \end{aligned}$$

(e) Suppose $P_1 = tau_s(P)$, then

$$\begin{aligned} lm_s(P_1, P_2) &= lm_s(tau_s(P), P_2) \\ &= tau_s(par_s(P, P_2)) \end{aligned}$$

10. $syn_s : Proc \times Proc \stackrel{\cdot}{\to} Proc$, where on any object $s \in |\mathbf{Inj}|$:

$$syn_s : \quad Proc(s) \times Proc(s) \quad \to \quad Proc(s)$$

We define the symmetric synchronization function $syn_s(P_1, P_2)$ by cases:

(a) Suppose $P_1 = in_s(x_i, \eta : s \to Proc(s), Q)$ and $P_2 = out_s(x_j, x_k, P)$, then

$$syn_s(P_1, P_2) = \begin{cases} par_s(\eta(x_k), P) & \text{if } x_i = x_j \\ nil_s & \text{otherwise} \end{cases}$$

(b) Suppose $P_1 = in_s(x_i, \eta : s \to Proc(s), Q_1)$ and $P_2 = bout_s(x_j, Q_2)$, then

$$syn_s(P_1, P_2) = \begin{cases} res_s(par_{s+1}(Q_1, Q_2)) & \text{if } x_i = x_j \\ nil_s & \text{otherwise} \end{cases}$$

(c) Suppose P_1, and P_2 are processes other than the ones mentioned in the cases above, then:

$$syn_s(P_1, P_2) \;=\; nil_s$$

8. We must show that the natural transformations satisfy the given equations: The above natural transformations must satisfy

1. Restriction For all i, **R**i is included in our definition of res.
2. Summation For all i, **S**i follows from the properties of the μ natural transformation for the finite powerset monad.
3. Parallel **P1** is included in the definition of par.
4. Left Merge For all i, **LM**i is included in the definition of lm
5. Synchronization For all i, **SYN**i is included in the definition of syn.

Therefore, we have constructed a concrete model for the finite π-calculus. The above shows that there exists at least one model satisfying our suggested properties with which to model the finite π-calculus.

4.2.1 Full-Abstraction and the Concrete Model

We have discussed in Chapter 3, the notions of late-bisimulation, denoted by $\sim_l$, and late-equivalence, denoted by $\dot{\sim}_l$. These described behavioral equalities between finite π-calculus processes. We will see in this section that the closed interpretation of the finite π-calculus captures exactly the notion of late-bisimulation and furthermore, that the open interpretation of the finite π-calculus captures exactly the notion of late-equivalence. These are results which [24] and [9], also achieve for their models of the full π-calculus. In this section, we base the proofs mainly on results suggested by

[24] but restricted to the finite π-calculus. Therefore, we end this section by proving the following full-abstraction results:

$$P \sim_l Q \Leftrightarrow (\!|s \vdash P|\!) = (\!|s \vdash Q|\!) \tag{7}$$

$$P \dot{\sim}_l Q \Leftrightarrow [\![s \vdash P]\!] = [\![s \vdash Q]\!] \tag{8}$$

We begin with a result which describes the final form of a process under the closed interpretation.

Lemma 4.2.5 *Given the closed interpretation in our model we have that for* $P \in Proc_f^\pi$*:*

$$(\!|s \vdash P|\!) = sum_s(b_1, \ldots, b_m)$$

where $s = \{x_1, \ldots, x_n\}$ *and* $b_i \in \{in_s(x_i, \langle(\!|s \vdash Q_1[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash Q|\!)), out_s(x_i, x_j, (\!|s \vdash Q|\!)), bout_s(x_i, (\!|s+1 \vdash Q|\!)), tau_s((\!|s \vdash Q|\!)), \emptyset\}$ *, for* $Q \in Proc_f^\pi$.

Proof. We proceed by structural induction on the process P.

(i) Let $P = 0$,

$$\begin{aligned} (\!|s \vdash 0|\!) &= \emptyset \\ \text{(by property of sum)} &= sum_s(\emptyset, \emptyset) \end{aligned}$$

Thus, the result holds.

(ii) Let $P = x(z).P'$,

$$\begin{aligned} (\!|s \vdash x_i(z).P'|\!) &= in_s(x_i, \langle(\!|s \vdash P'[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash P'|\!)) \\ \text{(by property of sum)} &= sum_s(in_s(x_i, \langle(\!|s \vdash P'[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash P'|\!)), \\ &\quad in_s(x_i, \langle(\!|s \vdash P'[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash P'|\!))) \end{aligned}$$

Thus, the result holds.

$(iii) - (v)$ Let $P = \overline{x_i}\langle x_j\rangle.P'$, $P = \overline{x_i}(z).P'$ or $P = \tau.P'$, the results hold by similar arguments made in (ii).

(*vi*) Let $P = \sum_{i\in I} \pi_i.P_i$, and by induction hypothesis $(\!|s \vdash \pi_i.P_i|\!) = sum_s(b_{(i,1)}, \ldots, b_{(i,m_i)})$.

$$\begin{aligned}
(\!|s \vdash \sum_{i\in I} \pi_i.P_i|\!) &= sum_s((\!|s \vdash \pi_1.P_1|\!), \ldots, (\!|s \vdash \pi_{|I|}.P_{|I|}|\!)) \\
\text{(I.H.)} &= sum_s(sum_s(b_{(1,1)}, \ldots, b_{(1,m_1)}), \ldots, sum_s(b_{(|I|,1)}, \ldots, b_{(|I|,m_{|I|})})) \\
\text{(by property of sum)} &= sum_s(b_{(1,1)}, \ldots, b_{(1,m_1)}, \ldots, b_{(|I|,1)}, \ldots, b_{(|I|,m_{|I|})})
\end{aligned}$$

Thus, the result holds.

(*vii*) Let $P = \nu z P'$, and by induction hypothesis $(\!|s+1 \vdash P'|\!) = sum_{s+1}(b_1, \ldots, b_m)$.

$$\begin{aligned}
(\!|s \vdash \nu z P'|\!) &= res_s((\!|s+1 \vdash P'|\!)) \\
\text{(I.H.)} &= res_s(sum_{s+1}(b_1, \ldots, b_m)) \\
\text{(by property of res)} &= sum_s(res_s(b_1), \ldots, res_s(b_m))
\end{aligned}$$

We must show that $res_s(b)$ is of the desired form. Let $z = x_{n+1}$,

(*a*) Suppose $b = in_{s+1}(x_i, \langle(\!|s+1 \vdash Q[z' := x_j]|\!)\rangle_{j=1}^{n+1}, (\!|s+1+1_{z'} \vdash Q|\!))$,

$$\begin{aligned}
res_s(b) &= res_s(in_{s+1}(x_i, \langle(\!|s+1 \vdash Q[z' := x_j]|\!)\rangle_{j=1}^{n+1}, \\
&\qquad (\!|s+1+1_{z'} \vdash Q|\!))) \\
\text{(by property of res)} &= \begin{cases} \{\emptyset\} & \text{if } x_i = z \\ in_s(x_i, \langle(\!|s \vdash (\nu z Q)[z' := x_j]|\!)\rangle_{j=1}^{n}, & \\ (\!|s+1_{z'} \vdash \nu z Q|\!)) & \text{else} \end{cases}
\end{aligned}$$

(*b*) Suppose $b = out_{s+1}(x_i, x_j, (\!|s+1 \vdash Q|\!))$

$$\begin{aligned}
res_s(b) &= res_s(out_{s+1}(x_i, x_j, (\!|s+1 \vdash Q|\!))) \\
\text{(by property of res)} &= \begin{cases} \{\emptyset\} & \text{if } x_i = z \\ bout_s(x_i, (\!|s+1 \vdash Q|\!)) & \text{if } x_i \neq z \text{ and } x_j = z \\ out_s(x_i, x_j, (\!|s \vdash \nu z Q|\!)) & \text{else} \end{cases}
\end{aligned}$$

(*c*) Suppose $b = bout_{s+1}(x_i, (\!|s+1+1_{z'} \vdash Q|\!))$

$$\begin{aligned}
res_s(b) &= res_s(bout_{s+1}(x_i, (\!|s+1+1_{z'} \vdash Q|\!))) \\
\text{(by property of res)} &= \begin{cases} \{\emptyset\} & \text{if } x_i = z \\ bout_s(x_i, (\!|s+1_{z'} \vdash \nu z Q|\!)) & \text{else} \end{cases}
\end{aligned}$$

(*d*) Suppose $b = tau_{s+1}(\llbracket s+1 \vdash Q \rrbracket)$

$$\begin{aligned} res_s(b) &= res_s(tau_{s+1}(\llbracket s+1 \vdash Q \rrbracket)) \\ \text{(by property of res)} &= tau_s(\llbracket s \vdash \nu z Q \rrbracket) \end{aligned}$$

(*e*) Suppose $b = \{\emptyset\}$

$$\begin{aligned} res_s(b) &= res_s(\{\emptyset\}) \\ \text{(by property of res)} &= \{\emptyset\} \end{aligned}$$

Thus, the result holds.

(*viii*) Suppose $P = P'|P''$, and by induction hypothesis that $\llbracket s \vdash P' \rrbracket = sum_s(b_1, \ldots, b_m)$ and $\llbracket s \vdash P'' \rrbracket = sum_s(c_1, \ldots, c_k)$.

$$\begin{aligned} \llbracket s \vdash P'|P'' \rrbracket &= par_s(\llbracket s \vdash P' \rrbracket, \llbracket s \vdash P'' \rrbracket) \\ &= sum_s(lm_s(\llbracket s \vdash P' \rrbracket, \llbracket s \vdash P'' \rrbracket), lm_s(\llbracket s \vdash P'' \rrbracket, \llbracket s \vdash P' \rrbracket), \\ &\quad syn_s(\llbracket s \vdash P' \rrbracket, \llbracket s \vdash P'' \rrbracket)) \\ \text{(I.H.)} &= sum_s(lm_s(sum_s(b_1, \ldots, b_m), \llbracket s \vdash P'' \rrbracket), \\ &\quad lm_s(sum_s(c_1, \ldots, c_k), \llbracket s \vdash P' \rrbracket), \\ &\quad syn_s(sum_s(b_1, \ldots, b_m), sum_s(c_1, \ldots, c_k))) \\ \text{(by property of lm and syn)} &= sum_s(\langle lm_s(b_u, \llbracket s \vdash P'' \rrbracket)\rangle_{u=1}^{m}, \langle lm_s(c_v, \llbracket s \vdash P' \rrbracket)\rangle_{v=1}^{k}, \\ &\quad \langle\langle syn_s(b_u, c_v)\rangle_{u=1}^{m}\rangle_{v=1}^{k}) \end{aligned}$$

Thus, if $a \in \llbracket s \vdash P'|P'' \rrbracket$ then $a \in \langle lm_s(b_u, \llbracket s \vdash P'' \rrbracket)\rangle_{u=1}^{m}$, $a \in \langle lm_s(c_v, \llbracket s \vdash P' \rrbracket)\rangle_{v=1}^{k}$, or $a \in \langle\langle syn_s(b_u, c_v)\rangle_{u=1}^{m}\rangle_{v=1}^{k}$. We verify each case,
(*a*) Suppose $a \in \langle lm_s(b_u, \llbracket s \vdash P'' \rrbracket)\rangle_{u=1}^{m}$,

$$lm_s(b_u, \llbracket s \vdash P'' \rrbracket)$$

$$= \begin{cases} (i) & in_s(x_i, \langle (\!|s \vdash (Q|P'')[z := x_j]|\!) \rangle_{j=1}^n, \\ & (\!|s+1 \vdash Q|P''|\!) \end{cases} \begin{array}{l} \text{if } b_u = \\ in_s(x_i, \langle (\!|s \vdash Q[z := x_j]|\!) \rangle_{j=1}^n,) \\ (\!|s+1 \vdash Q|\!) \end{array}$$

$$\begin{cases} (ii) & out_s(x_i, x_j, (\!|s \vdash Q|P''|\!)) & \text{if } b_u = out_s(x_i, x_j, (\!|s \vdash Q|\!)) \\ (iii) & bout_s(x_i, (\!|s+1 \vdash Q|P''|\!)) & \text{if } b_u = bout_s(x_i, (\!|s+1 \vdash Q|\!)) \\ (iv) & tau_s((\!|s \vdash Q|P''|\!)) & \text{if } b_u = tau_s((\!|s \vdash Q|\!)) \\ (v) & \{\emptyset\} & \text{if } b_u = \emptyset \end{cases}$$

(b) Suppose $a \in \langle lm_s(c_v, (\!|s \vdash P'|\!)) \rangle_{v=1}^k$, proceed with similar arguments made in (a).

(c) Suppose $a \in \langle \langle syn_s(b_u, c_v) \rangle_{u=1}^m \rangle_{v=1}^k$

$$syn_s(b_u, c_v)$$

$$= \begin{cases} (i) & tau_s((\!|s \vdash Q[z := x_j]|Q'|\!)) & \text{if } b_u = in_s(x_i, \langle (\!|s \vdash Q[z := x_j]|\!) \rangle_{j=1}^n, \\ & & (\!|s+1 \vdash Q|\!)) \text{ and } c_v = out_s(x_i, x_k, (\!|s \vdash Q'|\!)) \\ (ii) & tau_s((\!|s \vdash \nu z(Q|Q')|\!)) & \text{if } b_u = in_s(x_i, \langle (\!|s \vdash Q[z := x_j]|\!) \rangle_{j=1}^n, \\ & & (\!|s+1 \vdash Q|\!)) \text{ and } c_v = bout_s(x_i, (\!|s+1 \vdash Q'|\!)) \\ (iii) & syn_s(c_v, b_u) & \text{(by property of syn)} \\ (iv) & \{\emptyset\} & \text{else} \end{cases}$$

Thus, the result holds. □

In order to prove (7), we shall consider the relation between the elements of the closed interpretation $(\!|s \vdash P|\!)$ and the possible transitions of the process P. We will prove that the closed interpretation preserves and reflects these process transitions. Thus, we shall be able to prove our full abstraction result for the closed interpretation.

Lemma 4.2.6 *Let $P \in Proc_f^\pi$ be a finite π-calculus process then its closed interpretation in* $\mathbf{Set}^{\mathbf{Inj}}$ *reflects and preserves transitions.*

Definition 4.2.7 In order to prove the above lemma, we introduce two relations on the elements of $Proc(s)$ and $A_s^\pi \subseteq Proc_f^\pi$, where $A_s^\pi = \{P; fn(P) \subseteq s\}$. Let $p \in Proc(s)$ and $P \in A_s^\pi$,

(a) The relation $\triangleright_s \subseteq Proc(s) \times A_s^\pi$ is defined such that $p \triangleright_s P$ if and only if whenever

$$a = \left\{ \begin{array}{l} in_s(x_i, \langle (\!| s \vdash Q[z := x_j] |\!) \rangle_{j=1}^n, \\ (\!| s+1 \vdash Q |\!)) \\ \\ out_s(x_i, x_j, (\!| s \vdash Q |\!)) \\ \\ bout_s(x_i, (\!| s+1 \vdash Q |\!)) \\ \\ tau_s((\!| s \vdash Q |\!)) \end{array} \right\} \subseteq p \text{ then } \exists R. \left\{ \begin{array}{ll} P \xrightarrow{x(z)} R & \text{and } (\!| s+1 \vdash R |\!) \\ & = (\!| s+1 \vdash Q |\!) \\ P \xrightarrow{\overline{x}(y)} R & \text{and } (\!| s \vdash R |\!) \\ & = (\!| s \vdash Q |\!) \\ P \xrightarrow{\overline{x}(z)} R & \text{and } (\!| s+1 \vdash R |\!) \\ & = (\!| s+1 \vdash Q |\!) \\ P \xrightarrow{\tau} R & \text{and } (\!| s \vdash R |\!) \\ & = (\!| s \vdash Q |\!) \end{array} \right.$$

(b) The relation $\triangleleft_s \subseteq A_s^\pi \times Proc(s)$ is defined such that $P \triangleleft_s p$ if and only if whenever

$$\left\{ \begin{array}{ll} (i) & P \xrightarrow{x_i(z)} R \\ (ii) & P \xrightarrow{\overline{x_i}(x_j)} R \\ (iii) & P \xrightarrow{\overline{x_i}(z)} R \\ (iv) & P \xrightarrow{\tau} R \end{array} \right\} \text{ then } \left\{ \begin{array}{lll} (i) & in_s(x_i, \langle (\!| s \vdash R[z := x_j] |\!) \rangle_{j=1}^n, & \\ & (\!| s+1 \vdash R |\!)) & \subseteq (\!| s+1 \vdash P |\!) \\ (ii) & out_s(x_i, x_j, (\!| s \vdash R |\!)) & \subseteq (\!| s \vdash P |\!) \\ (iii) & bout_s(x_i, (\!| s+1 \vdash R |\!)) & \subseteq (\!| s+1 \vdash P |\!) \\ (iv) & tau_s((\!| s \vdash R |\!)) & \subseteq (\!| s+1 \vdash P |\!) \end{array} \right.$$

Proof.

(*A*) To prove that the closed interpretation reflects transitions, it will be enough to show that $(\!| s \vdash P |\!) \triangleright_s P$. Thus, since $(\!| s \vdash P |\!) = sum_s(b_1, \ldots, b_m)$ by Lemma 4.2.5, we must satisfy the following statement,

$$\text{if } b_u = \left\{ \begin{array}{l} in_s(x_i, \langle (\!| s \vdash Q[z := x_j] |\!) \rangle_{j=1}^n, \\ (\!| s+1 \vdash Q |\!)) \\ \\ out_s(x_i, x_j, (\!| s \vdash Q |\!)) \\ \\ bout_s(x_i, (\!| s+1 \vdash Q |\!)) \\ \\ tau_s((\!| s \vdash Q |\!)) \end{array} \right\} \text{ then } \exists R. \left\{ \begin{array}{ll} P \xrightarrow{x(z)} R & \text{and } (\!| s+1 \vdash R |\!) \\ & = (\!| s+1 \vdash Q |\!) \\ P \xrightarrow{\overline{x}(y)} R & \text{and } (\!| s \vdash R |\!) \\ & = (\!| s \vdash Q |\!) \\ P \xrightarrow{\overline{x}(z)} R & \text{and } (\!| s+1 \vdash R |\!) \\ & = (\!| s+1 \vdash Q |\!) \\ P \xrightarrow{\tau} R & \text{and } (\!| s \vdash R |\!) \\ & = (\!| s \vdash Q |\!) \end{array} \right.$$

We shall proceed by structural induction on finite π-calculus processes:

(i) Suppose $P = 0$.

Since $(\!|s \vdash 0|\!) = \emptyset$, there are no subsets of $(\!|s \vdash 0|\!)$ of the required form. Thus, $(\!|s \vdash 0|\!) \rhd_s 0$.

(ii) Suppose $P = x_i(z).P'$ and by induction hypothesis $(\!|s+1 \vdash P'|\!) \rhd_{s+1} P'$.

$$\begin{aligned}(\!|s \vdash x_i(z).P'|\!) &= in_s(x_i, \langle (\!|s \vdash P'[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash P'|\!)) \\ &= sum_s(in_s(x_i, \langle (\!|s \vdash P'[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash P'|\!)), \\ &\quad in_s(x_i, \langle (\!|s \vdash P'[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash P'|\!)))\end{aligned}$$

The only element of the required form is $in_s(x_i, \langle (\!|s \vdash P'[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash P'|\!))$ Hence, pick $R = P'$, we observe that $P \xrightarrow{x_i(z)} R$ and $(\!|s+1 \vdash R|\!) = (\!|s+1 \vdash P'|\!)$. Thus, we have satisfied the conditions of the relation.

$(iii) - (v)$ Suppose $P = \overline{x_i}\langle x_j\rangle.P'$, $P = \overline{x_i}(z).P'$ or $P = \tau.P'$ with their corresponding induction hypothesis, then the relation holds by similar arguments made in (ii).

(vi) Suppose $P = \sum_{i\in I} \pi_i.P_i$ and by induction hypothesis that $(\!|s \vdash \pi_i.P_i|\!) \rhd_s \pi_i.P_i$.

$$\begin{aligned}(\!|s \vdash \sum_{i\in I} \pi_i.P_i|\!) &= sum_s((\!|s \vdash \pi_1.P_1|\!), \ldots, (\!|s \vdash \pi_{|I|}.P_{|I|}|\!)) \\ \text{(by Lemma 4.2.5)} &= sum_s(sum_s(b_{(1,1)}, \ldots, b_{(1,m_1)}), \ldots, sum_s(b_{(|I|,1)}, \ldots, b_{(|I|,m_{|I|})})) \\ &= sum_s(b_{(1,1)}, \ldots, b_{(1,m_1)}, \ldots, b_{(|I|,1)}, \ldots, b_{(|I|,m_{|I|})}))\end{aligned}$$

Thus, by induction hypothesis we have satisfied the statement of the relation.

(vii) Suppose $P = \nu z P'$ and by induction hypothesis $(\!|s+1 \vdash P'|\!) \rhd_{s+1} P'$.

$$\begin{aligned}(\!|s \vdash \nu z P'|\!) &= res_s((\!|s+1 \vdash P'|\!)) \\ \text{(by Lemma 4.2.5)} &= res_s(sum_{s+1}(b_1, \ldots, b_m)) \\ &= sum_s(res_s(b_1), \ldots, res_s(b_m))\end{aligned}$$

We verify the validity of our statement for each possible form of $a = res_s(b)$.

(a) Suppose $b = in_{s+1}(x_i, \langle (\!| s+1 \vdash Q[z' := x_j] |\!) \rangle_{j=1}^{n+1}, (\!| s + 1_{z'} + 1 \vdash Q |\!))$,

$$res(b) = in_s(x_i, \langle (\!| s \vdash (\nu z Q)[z' := x_j] |\!) \rangle_{j=1}^{n}, (\!| s + 1_{z'} \vdash \nu z Q |\!))$$

By induction hypothesis, $\exists R'.\ P' \xrightarrow{x_i(z')} R'$ and $(\!| s + 1_{z'} + 1 \vdash R' |\!) = (\!| s + 1_{z'} + 1 \vdash Q |\!)$. Thus, let $R = \nu z R'$, we have that $P \xrightarrow{x_i(z')} R$ and that $(\!| s + 1_{z'} \vdash R |\!) = (\!| s + 1_{z'} \vdash \nu z R' |\!) = res_{s+1_{z'}}((\!| s + 1_{z'} + 1 \vdash R' |\!)) = res_{s+1_{z'}}((\!| s + 1_{z'} + 1 \vdash Q |\!)) = (\!| s + 1_{z'} \vdash \nu z Q |\!)$.

(b) Suppose $b = out_{s+1}(x_i, x_j, (\!| s + 1 \vdash Q |\!))$, by induction hypothesis, $\exists R'$ such that $P' \xrightarrow{\overline{x_i}\langle x_j \rangle} R'$ and $(\!| s \vdash R' |\!) = (\!| s \vdash Q |\!)$. We have that $res(b)$ can be of two forms:

(1) Suppose $x_j \neq z$

$$res(b) \;=\; out_s(x_i, x_j, (\!| s \vdash \nu z Q |\!))$$

Pick $R = \nu z R'$, then $P \xrightarrow{\overline{x_i}\langle x_j \rangle} R$, and $(\!| s \vdash R |\!) = (\!| s+1 \vdash \nu z R' |\!) = (\!| s+1 \vdash \nu z Q |\!)$.

(2) Suppose $x_j = z$

$$res(b) \;=\; bout_s(x_i, (\!| s + 1 \vdash Q |\!))$$

Pick $R = R'$, then $P \xrightarrow{\overline{x_i}(z)} R$, and $(\!| s + 1 \vdash R |\!) = (\!| s + 1 \vdash R' |\!) = (\!| s + 1 \vdash Q |\!)$.

(c) Suppose $b = bout_{s+1}(x, (\!| s + 1_{z'} + 1 \vdash Q |\!))$, by induction hypothesis there exists R' such that $P' \xrightarrow{\overline{x_i}(z)} R'$ and $(\!| s + 1_{z'} + 1 \vdash R' |\!) = (\!| s + 1_{z'} + 1 \vdash Q |\!)$.

$$res(b) \;=\; bout_s(x_i, (\!| s + 1_{z'} \vdash \nu z Q |\!))$$

pick $R = \nu z R'$, then $P \xrightarrow{\overline{x_i}(z)} R$, and $(\!| s + 1 \vdash R |\!) = (\!| s + 1 \vdash \nu z R' |\!) = (\!| s + 1 \vdash \nu z Q |\!)$.

(d) Suppose $b = tau_{s+1}((\!| s + 1 \vdash Q |\!))$, by induction hypothesis there exists R' such that $P' \xrightarrow{\tau} R'$ and $(\!| s + 1 \vdash R' |\!) = (\!| s + 1 \vdash Q |\!)$.

$$res(b) \;=\; tau_s((\!| s + 1 \vdash \nu z Q |\!))$$

Pick $R = \nu z R'$, then $P \xrightarrow{\tau} R$, and $(\!| s \vdash R |\!) = (\!| s \vdash \nu z R' |\!) = (\!| s \vdash \nu z Q |\!)$.

(*viii*) Suppose $P = P'|P''$ and by induction hypothesis that $(\!|s \vdash P'|\!) \triangleright_s P'$ and $(\!|s \vdash P''|\!) \triangleright_s P''$.

$$\begin{aligned}
(\!|s \vdash P'|P''|\!) &= par_s((\!|s \vdash P'|\!), (\!|s \vdash P''|\!)) \\
&= sum_s(lm_s((\!|s \vdash P'|\!), (\!|s \vdash P''|\!)), \\
&\quad lm_s((\!|s \vdash P''|\!), (\!|s \vdash P'|\!)), \\
&\quad syn_s((\!|s \vdash P'|\!), (\!|s \vdash P''|\!))) \\
\text{(by Lemma 4.2.5)} &= sum_s(lm_s(sum_s(b_1, \ldots, b_m), (\!|s \vdash P''|\!)), \\
&\quad lm_s(sum_s(c_1, \ldots, c_k), (\!|s \vdash P'|\!)), \\
&\quad syn_s(sum_s(b_1, \ldots, b_m), sum_s(c_1, \ldots, c_k))) \\
\text{(by property of lm and syn)} &= sum_s(\langle lm_s(b_u, (\!|s \vdash P''|\!))\rangle_{u=1}^m, \langle lm_s(c_v, (\!|s \vdash P'|\!))\rangle_{v=1}^k, \\
&\quad \langle\langle syn_s(b_u, c_v)\rangle_{u=1}^m\rangle_{v=1}^k)
\end{aligned}$$

We verify our statement for each valid form of the elements of the summation.

(a) Suppose $a = lm_s(b, (\!|s \vdash P''|\!))$. We want to show that a is of the required form as seen in Definition 4.2.7. There are four cases to verify

(1) Suppose $b = in_s(x_i, \langle(\!|s \vdash Q[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash Q|\!)) \in (\!|s \vdash P'|\!)$. By induction hypothesis, there exists R' such that $P' \xrightarrow{x_i(z)} R'$ and $(\!|s+1 \vdash R'|\!) = (\!|s+1 \vdash Q|\!)$.

$$a = in_s(x_i, \langle(\!|s \vdash (Q|P'')[z := x_j]|\!)\rangle_{j=1}^n, (\!|s+1 \vdash Q|P''|\!))$$

Pick $R = R'|P''$, we have that $P \xrightarrow{x_i(z)} R$ and that $(\!|s+1 \vdash R|\!) = par_s((\!|s+1 \vdash R'|\!), (\!|s+1 \vdash P''|\!)) = (\!|s+1 \vdash Q|P''|\!)$.

(2)-(4) The result for $b = out_s(x_i, x_j, (\!|s \vdash Q|\!), b = bout_s(x_i, (\!|s+1 \vdash Q|\!))$ or $b = tau_s((\!|s \vdash Q|\!))$ with the corresponding induction hypothesis, arise from similar arguments made in (1).

(b) Suppose $a \in lm_s(c, (\!|s \vdash P'|\!))$, the result arises by similar arguments discussed in (a).

(c) Suppose $a \in syn_s(b,c)$, there are two cases to consider

(1) Suppose $b = in_s(x_i, \langle (\!| s \vdash Q[z := x_j] |\!) \rangle_{j=1}^n, (\!| s+1 \vdash Q |\!))$ and $c = out_s(x_i, x_k, (\!| s \vdash Q' |\!))$. By induction hypothesis, there exists R_1' and R_2' such that $P' \xrightarrow{x_i(z)} R_1'$, $P'' \xrightarrow{\overline{x_i}(x_k)} R_2'$, $(\!| s+1 \vdash R_1' |\!) = (\!| s+1 \vdash Q |\!)$ and $(\!| s \vdash R_2' |\!) = (\!| s \vdash Q' |\!)$.

$$a \;=\; tau_s((\!| s \vdash Q[z := x_j] | Q' |\!)$$

Pick $R = R_1'[z := x_k] | R_2'$, we have that $P \xrightarrow{\tau} R$ and that $(\!| s \vdash R |\!) = par_s((\!| s \vdash R_1'[z := x_k] |\!), (\!| s \vdash R_2' |\!)) = par_s((\!| s \vdash Q[z := x_k] |\!), (\!| s \vdash Q' |\!)) = (\!| s \vdash Q[z := x_k] | Q' |\!)$.

(2) Suppose $b = in_s(x_i, \langle (\!| s \vdash Q[z := x_j] |\!) \rangle_{j=1}^n, (\!| s+1 \vdash Q |\!))$ and $c = bout_s(x_i, (\!| s+1 \vdash Q' |\!))$. By induction hypothesis, there exists R_1' and R_2' such that $P' \xrightarrow{x_i(z)} R_1'$, $P'' \xrightarrow{\overline{x_i}(z)} R_2'$, $(\!| s+1 \vdash R_1' |\!) = (\!| s+1 \vdash Q |\!)$ and $(\!| s+1 \vdash R_2' |\!) = (\!| s+1 \vdash Q' |\!)$.

$$a \;=\; tau_s((\!| s \vdash \nu z(Q | Q') |\!)$$

Pick $R = \nu z(R_1' | R_2')$, we have that $P \xrightarrow{\tau} R$ and that $(\!| s \vdash R |\!) = res_s(par_{s_1}((\!| s+1 \vdash R_1' |\!), (\!| s+1 \vdash R_2' |\!))) = res_s(par_s((\!| s+1 \vdash Q |\!), (\!| s \vdash Q' |\!))) = (\!| s \vdash \nu z(Q | Q') |\!)$.

(B) To prove that the closed interpretation preserves transitions, it will be enough to show that $P \lhd_s (\!| s \vdash P |\!)$. Thus, since $(\!| s \vdash P |\!) = sum_s(b_1, \ldots, b_m)$ by Lemma 4.2.5, we must satisfy the following statement,

$$\left.\begin{array}{ll} (i) & P \xrightarrow{x_i(z)} R \\ (ii) & P \xrightarrow{\overline{x_i}(x_j)} R \\ (iii) & P \xrightarrow{\overline{x_i}(z)} R \\ (iv) & P \xrightarrow{\tau} R \end{array}\right\} \text{ then } \exists u.\, b_u = \left\{\begin{array}{ll} (i) & in_s(x_i, \langle (\!| s \vdash R[z := x_j] |\!) \rangle_{j=1}^n, (\!| s+1 \vdash R |\!)) \\ (ii) & out_s(x_i, x_j, (\!| s \vdash R |\!)) \\ (iii) & bout_s(x_i, (\!| s+1 \vdash R |\!)) \\ (iv) & tau_s((\!| s \vdash R |\!)) \end{array}\right.$$

We proceed by induction on the last rule of the derivation of the transition.

(i) Suppose $P \stackrel{x_i(z)}{\rightarrow} R$, there are four possible derivations to check:

(a) Suppose $P = x_i(z).P'$ and $R = P'$.

$$\begin{aligned}
(\!|s \vdash P|\!) &= (\!|s \vdash x_i(z).P'|\!) \\
&= in_s(x_i, \langle (\!|s \vdash P'[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash P'|\!)) \\
&= in_s(x_i, \langle (\!|s \vdash R[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash R|\!)) \\
&= sum_s(in_s(x_i, \langle (\!|s \vdash P'[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash P'|\!)), \\
&\quad in_s(x_i, \langle (\!|s \vdash P'[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash P'|\!)))
\end{aligned}$$

Thus, the relation holds.

(b) Suppose $P = P' + Q$, $R = P''$ and $P' \stackrel{x_i(z)}{\rightarrow} P''$. By induction hypothesis, we have that $P' \lhd_s (\!|s \vdash P'|\!)$. Therefore, since $(\!|s \vdash P'|\!) = sum_s(b_1, \ldots, b_m)$, it must be the case that for some $1 \leq u_0 \leq m$, $b_{u_0} = in_s(x_i, \langle (\!|s \vdash P''[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash P''|\!))$.

$$\begin{aligned}
(\!|s \vdash P|\!) &= (\!|s \vdash P' + Q|\!) \\
\text{(by Lemma 4.2.5)} &= sum_s(b_1, \ldots, b_m, c_1, \ldots, c_k)
\end{aligned}$$

Thus, pick $u = u_0$ and the relation holds.

(c) Suppose $P = P'|Q$, $R = P''|Q$, $z \notin fn(Q)$ and $P' \stackrel{x_i(z)}{\rightarrow} P''$. By induction hypothesis, we have that $P' \lhd_s (\!|s \vdash P'|\!)$. Therefore, since $(\!|s \vdash P'|\!) = sum_s(b_1, \ldots, b_m)$, it must be the case that for some $1 \leq u_0 \leq m$, $b_{u_0} = in_s(x_i, \langle (\!|s \vdash P''[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash P''|\!))$.

$$\begin{aligned}
(\!|s \vdash P|\!) &= (\!|s \vdash P'|Q|\!) \\
\text{(by Lemma 4.2.5)} &= sum_s(\langle lm_s(b_u, (\!|s \vdash Q|\!)) \rangle_{u=1}^m, \langle lm_s(c_v, (\!|s \vdash P'|\!)) \rangle_{v=1}^k, \\
&\quad \langle \langle syn_s(b_u, c_v) \rangle_{u=1}^n \rangle_{v=1}^k)
\end{aligned}$$

Thus, pick $lm_s(b_{u_0}, (\!|s \vdash Q|\!)) = in_s(x_i, \langle (\!|s \vdash (P''|Q)[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash P''|Q|\!))$ and the relation holds.

(d) Suppose $P = \nu z' P'$, $R = \nu z' P''$, $P' \stackrel{x_i(z)}{\rightarrow} P''$ and $z' \neq x_i, z$. By induction hypothesis, we have that $P' \lhd_s (\!|s \vdash P'|\!)$. Therefore, since $(\!|s \vdash P'|\!) = sum_s(b_1, \ldots, b_m)$, it must be the case that for some $1 \leq u_0 \leq m$, $b_{u_0} = in_s(x_i, \langle (\!|s \vdash P''[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash P''|\!))$.

$$\begin{aligned} (\!|s \vdash P|\!) &= (\!|s \vdash \nu z' P'|\!) \\ \text{(by Lemma 4.2.5)} &= sum_s(res_s(b_1), \ldots, res_s(b_m)) \end{aligned}$$

Thus, pick $res_s(b_{v_0}) = in_s(x_i, \rangle (\!|s \vdash (\nu z' P'')[z := x_j]|\!) \rangle_{j=1}^n, (\!|s+1 \vdash \nu z' P''|\!))$ and the relation is satisfied.

(ii) Suppose $P \stackrel{\overline{x_i}(x_j)}{\rightarrow} R$, which satisfies the relation by similar arguments made in (i).

(iii) Suppose $P \stackrel{\overline{x_i}(z)}{\rightarrow} R$, there are five possible derivations to check:

(a), (b), (c), and (d) arise by similar arguments made in (i).

(e) Suppose $P = \nu z P'$, $R = P''$, $P' \stackrel{\overline{x_i}(z)}{\rightarrow} P''$ and $x_i \neq z$. By induction hypothesis, we have that $P' \lhd_s (\!|s \vdash P'|\!)$. Therefore, since $(\!|s \vdash P'|\!) = sum_s(b_1, \ldots, b_m)$, it must be the case that for some $1 \leq u_0 \leq m$, $b_{u_0} = out_s(x_i, z, (\!|s \vdash P''|\!))$.

$$\begin{aligned} (\!|s \vdash P|\!) &= (\!|s \vdash \nu z P'|\!) \\ \text{(by Lemma 4.2.5)} &= sum_s(res_s(b_1), \ldots, res_s(b_m)) \end{aligned}$$

Thus, pick $res_s(b_{v_0}) = bout_s(x_i, (\!|s+1 \vdash P''|\!))$ and the relation is satisfied.

(iv) Suppose $P \stackrel{\tau}{\rightarrow} R$, there are six possible derivations to check:

(a), (b), (c), and (d) Arise from similar arguments made in (i)

(e) Suppose $P = P'|Q$, $R = P''[z := x_j]|Q'$, $P' \stackrel{x_i(z)}{\rightarrow} P''$ and $Q \stackrel{\overline{x_i}(x_j)}{\rightarrow} Q'$. By induction hypothesis, we have that $P' \lhd_s (\!|s \vdash P'|\!)$ and $Q \lhd_s (\!|s \vdash Q|\!)$. Therefore, since $(\!|s \vdash P'|\!) = sum_s(b_1, \ldots, b_m)$ and $(\!|s \vdash Q|\!) = sum_s(c_1, \ldots, c_k)$, it must be the case

that for some $1 \leq u_0, v_0 \leq m$, $b_{u_0} = in_s(x_i, \langle (\!| s \vdash P''[z := x_j] |\!) \rangle_{j=1}^n, (\!| s+1 \vdash P'' |\!))$ and $c_{v_0} = out_s(x_i, x_j, (\!| s \vdash Q' |\!))$.

$$\begin{aligned} (\!| s \vdash P |\!) &= (\!| s \vdash P'|Q |\!) \\ \text{(by Lemma 4.2.5)} &= sum_s(\langle lm_s(b_u, (\!| s \vdash Q |\!)) \rangle_{u=1}^m, \langle lm_s(c_v, (\!| s \vdash P' |\!)) \rangle_{v=1}^k, \\ &\quad \langle \langle syn_s(b_u, c_v) \rangle_{u=1}^n \rangle_{v=1}^k) \end{aligned}$$

Thus, pick $syn_s(b_{u_0}, c_{v_0}) = tau_s((\!| s \vdash (P''[z := x_f] | Q' |\!))$ and the relation holds.

(f) Suppose $P = P'|Q$, $R = \nu z(P''|Q')$, $P' \xrightarrow{x_i(z)} P''$ and $Q \xrightarrow{\overline{x_i}(z)} Q'$. By induction hypothesis, we have that $P' \lhd_s (\!| s \vdash P' |\!)$ and $Q \lhd_s (\!| s \vdash Q |\!)$. Therefore, since $(\!| s \vdash P' |\!) = sum_s(b_1, \ldots, b_m)$ and $(\!| s \vdash Q |\!) = sum_s(c_1, \ldots, c_k)$, it must be the case that for some $1 \leq u_0, v_0 \leq m$, $b_{u_0} = in_s(x_i, \langle (\!| s \vdash P''[z := x_j] |\!) \rangle_{j=1}^n, (\!| s+1 \vdash P'' |\!))$ and $c_{v_0} = bout_s(x_i, (\!| s \vdash Q' |\!))$.

$$\begin{aligned} (\!| s \vdash P |\!) &= (\!| s \vdash P'|Q |\!) \\ \text{(by Lemma 4.2.5)} &= sum_s(\langle lm_s(b_u, (\!| s \vdash Q |\!)) \rangle_{u=1}^m, \langle lm_s(c_v, (\!| s \vdash P' |\!)) \rangle_{v=1}^k, \\ &\quad \langle \langle syn_s(b_u, c_v) \rangle_{u=1}^n \rangle_{v=1}^k) \end{aligned}$$

Thus, pick $syn_s(b_{u_0}, c_{v_0}) = tau_s((\!| s \vdash \nu z(P''|Q') |\!))$ and the relation holds. □

Now we can prove the main theorem of this section.

Theorem 4.2.8 (Full Abstraction)

$$(\!| s \vdash P |\!) = (\!| s \vdash Q |\!) \Leftrightarrow P \sim_l Q$$

Proof. By Lemma 4.2.6, we know that $(\!| s \vdash Q |\!) \lhd_s \rhd_s Q$ and $(\!| s \vdash P |\!) \lhd_s \rhd_s P$.

($\Rightarrow$) Suppose $(\!| s \vdash Q |\!) = (\!| s \vdash P |\!) = sum_s(b_1, \ldots, b_m)$. Therefore, we have that for every $b_u \neq \{\emptyset\}$ there exists R and R' such that $P \xrightarrow{\pi} R$ and $Q \xrightarrow{\pi} R'$ and $(\!| s \vdash R |\!) = (\!| s \vdash R' |\!)$. So by induction hypothesis, every commitment that P can make via π can be duplicated by an identical π commitment by Q. Moreover, their

continuations are also in relation. Therefore, we get that $P \sim_l Q$.

($\Leftarrow$) Suppose $P \sim_l Q$. Therefore, for every transition $P \xrightarrow{\pi} R$ and $Q \xrightarrow{\pi} R'$ such that $R \sim_l R'$, there is an element b_u of the required form such that $(\!|s \vdash P|\!) = sum_s(b_1, \ldots, b_m) = (\!|s \vdash Q|\!)$. This satisfies the equality. □

With the full abstraction result for the closed interpretation and due to the closely formulated definitions of the open interpretation, we can easily show the full abstraction for the open interpretation and prove (8).

Corollary 4.2.9

$$[\![s \vdash P]\!] = [\![s \vdash Q]\!] \Leftrightarrow P \dot{\sim}_l Q$$

Proof. ($\Rightarrow$) Suppose $[\![s \vdash P]\!] = [\![s \vdash Q]\!]$, $s = \{x_1, \ldots, x_n\}$. Let σ be an arbitrary substitution and $s' = \{b_1, \ldots, b_m\}$ such that $\sigma(x_i) \in s'$ for all $1 \leq i \leq n$.

$$\begin{aligned}
[\![s' \vdash P\sigma]\!] &= [\![s' + s \vdash P]\!] \circ \langle [\![s' \vdash \sigma(x_1)]\!], \ldots, [\![s' \vdash \sigma(x_n)]\!] \rangle \\
&= [\![s \vdash P]\!] \circ \langle \pi_{|s'|+1}, \ldots, \pi_{|s'|+|s|} \rangle \circ \langle [\![s' \vdash \sigma(x_1)]\!], \ldots, [\![s' \vdash \sigma(x_n)]\!] \rangle \\
&= [\![s \vdash Q]\!] \circ \langle \pi_{|s'|+1}, \ldots, \pi_{|s'|+|s|} \rangle \circ \langle [\![s' \vdash \sigma(x_1)]\!], \ldots, [\![s' \vdash \sigma(x_n)]\!] \rangle \\
&= [\![s' + s \vdash Q]\!] \circ \langle [\![s' \vdash \sigma(x_1)]\!], \ldots, [\![s' \vdash \sigma(x_n)]\!] \rangle \\
&= [\![s' \vdash Q\sigma]\!]
\end{aligned}$$

Therefore using the Theorem 4.2.8, for an arbitrary substitution σ;

$$\begin{aligned}
[\![s \vdash P]\!] = [\![s \vdash Q]\!] &\Rightarrow [\![s' \vdash P\sigma]\!] = [\![s' \vdash Q\sigma]\!] \\
&\Rightarrow [\![s' \vdash P\sigma]\!]_{s'}(b_1, \ldots, b_m) = [\![s' \vdash Q\sigma]\!]_{s'}(b_1, \ldots, b_m) \\
&\Rightarrow (\!|s' \vdash P\sigma|\!) = (\!|s' \vdash Q\sigma|\!) \\
&\Rightarrow P\sigma \sim_l Q\sigma
\end{aligned}$$

Since σ was chosen arbitrarily, we get that $P \dot{\sim}_l Q$.
($\Leftarrow$) Suppose $P \dot{\sim}_l Q$,

$$P \dot{\sim}_l Q \Rightarrow P\sigma \sim Q\sigma, \text{ for all substitutions } \sigma$$

$$
\begin{aligned}
&\Rightarrow (\!|s' \vdash P\sigma|\!) = (\!|s' \vdash Q\sigma|\!), \text{by Theorem 4.2.8} \\
&\Rightarrow [\![s' \vdash P\sigma]\!]_{s'}(b_1, \ldots, b_m) = [\![s' \vdash Q\sigma]\!]_{s'}(b_1, \ldots, b_m) \\
&\Rightarrow [\![s' + s \vdash P]\!]_{s'} \circ \pi_S(b_1, \ldots, b_m) = [\![s' + s \vdash Q]\!]_{s'} \pi_S(b_1, \ldots, b_m) \\
&\Rightarrow [\![s \vdash P]\!]_{s'} \circ \pi_C \circ \pi_S(b_1, \ldots, b_m) = [\![s \vdash Q]\!]_{s'} \circ \pi_C \circ \pi_S(b_1, \ldots, b_m) \\
&\Rightarrow [\![s \vdash P]\!]_{s'} \circ \langle [\![s' \vdash b_1]\!], \ldots, [\![s' \vdash b_m]\!] \rangle_{s'}(b_1, \ldots, b_m) \\
&\quad = [\![s \vdash Q]\!]_{s'} \circ \langle [\![s' \vdash b_1]\!], \ldots, [\![s' \vdash b_m]\!] \rangle_{s'}(b_1, \ldots, b_m)
\end{aligned}
$$

Where $\pi_S = \langle \pi_1, \ldots, \pi_m, [\![s' \vdash b_1]\!], \ldots, [\![s' \vdash b_m]\!] \rangle_{s'}$ and $\pi_C = \langle \pi_{m+1}, \ldots, \pi_{m+n} \rangle_{s'}$. We must show that $[\![s \vdash P]\!]_t(t_1, \ldots, t_n) = [\![s \vdash Q]\!]_t(t_1, \ldots, t_n)$, for any $t \in |\mathbf{Inj}|$ and $(t_1, \ldots, t_n) \in N^{|s|}(t)$. Therefore, in the above calculations pick σ such that $\sigma(x_i) = t_i$ and $s' = t$. Thus we have shown that

$$[\![s \vdash P]\!]_t \circ (t_1, \ldots, t_m) \;=\; [\![s \vdash Q]\!]_t(t_1, \ldots, t_m)$$

□

Chapter 5

Conclusions

We can see from our full-abstraction theorems how our denotational approach is superior to an operational approach. We can actually study our late-bisimulation $\sim_l$ and late-equivalence $\dot{\sim}_l$ relation by analyzing the equality between objects in our model. The most prominent of these properties we have verified is the preservation of the $\dot{\sim}_l$ under substitution, which arose as a consequence of our open interpretation. Therefore, the construction of a denotational model is used to verify that the properties we wished in our language exist. Moreover, it is a useful tool in verifying how other properties might arise: we study closely the relation between equality in the model and late-bisimulation or late-equivalence. For example, we can easily convince ourselves of the validity of the restriction extrusion property i.e $\nu z(P|Q) \dot{\sim}_l P|\nu zQ$ if $z \notin fn(P)$. Similarly, we can also justify the associativity of the parallel composition operator.

A key point in the construction of our model arose in the definition of the object of processes

$$Proc \;=\; P_f^{\mathbf{Inj}}(H(Proc)) \tag{9}$$

where $H(X) = N \times (N \Rightarrow X) + N \times N \times X + N \times (N \multimap X) + X$ and $P_f^{\mathbf{Inj}}$ was the lifted finite powerset monad. This construction could be used to model other similar computation models by varying our chosen monad $P_f^{\mathbf{Inj}}$ and even modifying the endofunctor $H : \mathbf{Set}^{\mathbf{Inj}} \to \mathbf{Set}^{\mathbf{Inj}}$. For example, [9] admits the following:

- to deal with global variables one should replace $P_f^{\mathbf{Inj}}$ with the monad $TX = P_f^{\mathbf{Inj}}(X \times S)^S$, where S is an object of states.
- to deal with the π_I-calculus, one should use the endofunctor $HX = N \times (N \Rightarrow X) + N \times (N \multimap X) + X$. This gives a fully-abstract model for the π_I-bisimularity.

It's natural when considering expressions with free variables to view them as fibred categories (or hyperdoctrines) as seen in [15]. But the syntax of the π-calculus does not seem to lead to a hyperdoctrine structure. The reason it fails is due to the impossibility of the following inference:

$$\frac{\nu z(\overline{x}\langle z\rangle.P) \vdash \overline{x}(z).P}{\overline{x}\langle z\rangle.P \vdash \overline{x}(z).P}$$

where $A \vdash B$ means that B simulates A. It cannot occur, since $\overline{x}(z).P$ does not simulate $\overline{z}\langle x\rangle.P$. However presheaf categories suggest there is a fibred structure to study. Since our models are fully-abstract the model might suggest a variation of the syntax and/or the operational semantics which would lead to a hyperdoctrine. For example, it is known that the π_I-calculus satisfies the above existential quantifier adjointness property. [1]

Furthermore, it would be very interesting to find other presheaf categories which satisfy the properties that we defined. This would allow us to find a different model for the Finite π-calculus. For example, can we pick the presheaf category of G-Sets? However, the intuition behind the use of the category **Inj** is very convincing. So this may not be such an easy task.

[1] Proved by P. Panagaden, McGill, but not published.

Appendix A

Proof of Lemma 4.1.6

We begin by stating and proving a result we shall need in the proof of Lemma 4.1.6.

Lemma A.1 *Let λ be the currying operation in* $\mathbf{Set}^{\mathbf{Inj}}$ *and Λ be the operator defined in Lemma 4.1.4. Then for any $\psi : N^{|N(s)|+1} \stackrel{\cdot}{\to} Proc$ we have that $\lambda\psi : N^{|N(s)|} \stackrel{\cdot}{\to} (N \Rightarrow Proc)$ and $\Lambda\psi : N^{|N(s)|} \stackrel{\cdot}{\to} (N \multimap Proc)$. Hence, we get the following results:*

1. *For any $\psi : N^{|N(s)|+1} \stackrel{\cdot}{\to} Proc$, then $\lambda(\psi \circ \pi_P)$ is such that $\lambda(\psi \circ \pi_P) = \lambda(\psi) \circ \pi'_P$, where $\pi'_P = \langle \pi_1, \pi_2, \ldots, \pi_{|N(s)|}, \pi_{|N(s)|+2}, \pi_{|N(s)|+1}, \pi_{|N(s)|+3}, \ldots, \pi_{|N(s)|+|N(t)|+1} \rangle$. Similar results hold for π_S and π_C.*

2. *For any $\psi : N^{|N(s)|+1} \stackrel{\cdot}{\to} Proc$, then $\Lambda(\psi \circ \pi_P)$ is such that $\Lambda(\psi \circ \pi_P) = \Lambda(\psi) \circ \pi'_P$, where $\pi'_P = \langle \pi_1, \pi_2, \ldots, \pi_{|N(s)|}, \pi_{|N(s)|+2}, \pi_{|N(s)|+1}, \pi_{|N(s)|+3}, \ldots, \pi_{|N(s)|+|N(t)|+1} \rangle$. Similar results hold for π_S and π_C.*

Proof.

1. From [13], p.54 (3.2), we have the following equality in any ccc:

$$\lambda(h\langle k\pi_1, \pi_2 \rangle) = \lambda(h)k$$

   Hence let $h = \psi$, and $k = \pi'_P$. Therefore we obtain that:

$$\lambda(\psi) \circ \pi'_P = \lambda(\psi\langle \pi'_P \circ \pi, \pi' \rangle) = \lambda(\psi\langle \pi'_P, \pi_{|N(s)|+|N(t)|+2} \rangle) = \lambda(\psi \circ \pi_P)$$

2. By Lemma 4.1.4, $\Lambda = surj \circ \lambda$ hence it holds by 1.

□

Remark A.1 In the above Lemma for ease of notation, we shall often denote the result in the following way:

$$\lambda(\psi \circ \pi_P) = \lambda(\psi) \circ \pi_P$$

and not distinguish π_P and π'_P and let the situation dictate which we are referring to.

Now recall the result we wish to show.

Lemma A.2 *The following are four properties of the open interpretation. Let* $s, 1_a, 1_b, t \in |\mathbf{Inj}|$.

1. *Permutation*
   Let $N(s + 1_a + 1_b + t) \vdash P$, *then:*

$$\left[\!\left[N(s + 1_a + 1_b + t) \vdash P \right]\!\right] = \left[\!\left[N(s + 1_b + 1_a + t) \vdash P \right]\!\right] \circ \pi_P$$

   where $\pi_P = \langle \pi_1, \pi_2, \ldots, \pi_{|N(s)|}, \pi_{|N(s)|+2}, \pi_{|N(s)|+1}, \pi_{|N(s)|+3}, \ldots, \pi_{|N(s)|+|N(t)|+2} \rangle$

2. *Contraction*
   Let $N(s + t) \vdash P$, *and* $a' \in N(1_a)$, *then:*

$$\left[\!\left[N(s + 1_a + t) \vdash P \right]\!\right] = \left[\!\left[N(s + t) \vdash P \right]\!\right] \circ \pi_C$$

   where $\pi_C = \langle \pi_1, \pi_2, \ldots, \pi_{|N(s)|}, \pi_{|N(s)|+2}, \ldots, \pi_{|N(s)|+|N(t)|+1} \rangle$

3. *Substitution*
   Let $N(s + 1_a + t) \vdash P$, $a' \in N(1_a)$, *and* $x_j \in N(s + t)$, *then:*

$$\left[\!\left[N(s + t) \vdash P[a' := x_j] \right]\!\right] = \left[\!\left[N(s + 1_a + t) \vdash P \right]\!\right] \circ \pi_S$$

   where $\pi_S = \langle \pi_1, \pi_2, \ldots, \pi_{|N(s)|}, \left[\!\left[N(s + t) \vdash x_j \right]\!\right], \pi_{|N(s)|+1}, \ldots, \pi_{|N(s)|+|N(t)|} \rangle$

4. *Alpha-Conversion*
   Let $N(s) \vdash P$ *and* $N(s) \vdash Q$. *If* $P \equiv_\alpha Q$, *then:*

$$\left[\!\left[N(s) \vdash P \right]\!\right] = \left[\!\left[N(s) \vdash Q \right]\!\right]$$

Proof.

1. (PERMUTATION) Suppose $N(s+1_a+1_b+t) \vdash P$, we will show by structural induction on P that:

$$\left[\!\left[N(s+1_a+1_b+t) \vdash P \right]\!\right] = \left[\!\left[N(s+1_b+1_a+t) \vdash P \right]\!\right] \circ \pi_P$$

Let,

- $N(s+1_a+1_b+t) = \{x_1, x_2, \ldots, x_{|N(s)|}, a' = x_{|N(s)|+1}, b' = x_{|N(s)|+2}, x_{|N(s)|+3}, \ldots, x_{|N(s)|+|N(t)|+2}\}$,
- $z \in N(1)$, and
- **Induction Hypothesis:**
  $\left[\!\left[N(s+1_a+1_b+t) \vdash P_i \right]\!\right] = \left[\!\left[N(s+1_b+1_a+t) \vdash P_i \right]\!\right] \circ \pi_P$ for $i = 1, 2$.

(i) We begin with verifying that the equation works on our interpretation of names. Let $x_i \in N(s+1_a+1_b+t)$, $s' \in |\mathbf{Inj}|$, and $y_j \in N(s'), 1 \leq j \leq |N(s)| + |N(t)| + 2$:

$$\begin{aligned}
\text{LHS} &= \left[\!\left[N(s+1_a+1_b+t) \vdash x_i \right]\!\right]_{s'}(y_1, \ldots, y_{|N(s)|+|N(t)|+2}) \\
&= \pi_i(y_1, \ldots, y_{|N(s)|+|N(t)|+2}) \\
&= y_i \\
\text{RHS} &= \left[\!\left[N(s+1_b+1_a+t) \vdash x_i \right]\!\right]_{s'} \circ \pi_P(y_1, \ldots, y_{|N(s)|+|N(t)|+2}) \\
&= \begin{cases}
\pi_{|N(s)|+1}(y_1, \ldots, y_{|N(s)|}, y_{|N(s)|+2}, y_{|N(s)|+1}, y_{|N(s)|+3}, \\
\ldots, y_{|N(s)|+|N(t)|+2}) & \text{if } x_i = b' \\
\pi_{|N(s)|+2}(y_1, \ldots, y_{|N(s)|}, y_{|N(s)|+2}, y_{|N(s)|+1}, y_{|N(s)|+3}, \\
\ldots, y_{|N(s)|+|N(t)|+2}) & \text{if } x_i = a' \\
\pi_i(y_1, \ldots, y_{|N(s)|}, y_{|N(s)|+2}, y_{|N(s)|+1}, y_{|N(s)|+3}, \\
\ldots, y_{|N(s)|+|N(t)|+2}) & \text{else}
\end{cases} \\
&= y_i
\end{aligned}$$

We have that $RHS = LHS$

(*ii*) Suppose $P = 0$. Let $s' \in |\mathbf{Inj}|$, and $y_j \in N(s')$, for $1 \leq j \leq |N(s)| + |N(t)| + 2$:

$$\begin{aligned}
\text{LHS} &= \left[\!\left[N(s + 1_a + 1_b + t) \vdash 0 \right]\!\right]_{s'}(y_1, \ldots, y_{|N(s)|+|N(t)|+2}) \\
&= nil_{s+1_a+1_b+t} \\
\text{RHS} &= \left[\!\left[N(s + 1_b + 1_a + t) \vdash 0 \right]\!\right]_{s'} \circ \pi_P(y_1, \ldots, y_{|N(s)|+|N(t)|+2}) \\
&= \left[\!\left[N(s + 1_b + 1_a + t) \vdash 0 \right]\!\right]_{s'}(y_1, \ldots, y_{|N(s)|}, y_{|N(s)|+2}, \\
&\quad y_{|N(s)|+1}, y_{|N(s)|+3}, \ldots, y_{|N(s)|+|N(t)|+2}) \\
&= nil_{s+1_b+1_a+t}
\end{aligned}$$

Therefore, RHS = LHS.

(*iii*) Suppose $P = x_i(z).P_1$

$$\begin{aligned}
&\left[\!\left[N(s + 1_a + 1_b + t) \vdash x_i(z).P_1 \right]\!\right] \\
= \;& in \circ \langle \left[\!\left[N(s + 1_a + 1_b + t) \vdash x_i \right]\!\right], \lambda \left[\!\left[N(s + 1_a + 1_b + t + 1) \vdash P_1 \right]\!\right] \rangle \\
= \;& in \circ \langle \left[\!\left[N(s + 1_b + 1_a + t) \vdash x_i \right]\!\right] \circ \pi_P, \\
& \lambda(\left[\!\left[N(s + 1_b + 1_a + t + 1) \vdash P_1 \right]\!\right] \circ \pi_P) \rangle \\
= \;& in \circ \langle \left[\!\left[N(s + 1_b + 1_a + t) \vdash x_i \right]\!\right] \circ \pi_P, \\
& \lambda(\left[\!\left[N(s + 1_b + 1_a + t + 1) \vdash P_1 \right]\!\right]) \circ \pi_P \rangle \\
= \;& in \circ \langle \left[\!\left[N(s + 1_b + 1_a + t) \vdash x_i \right]\!\right], \lambda \left[\!\left[N(s + 1_b + 1_a + t + 1) \vdash P_1 \right]\!\right] \rangle \circ \pi_P \\
= \;& \left[\!\left[N(s + 1_b + 1_a + t) \vdash x_i(z).P_1 \right]\!\right] \circ \pi_P
\end{aligned}$$

(*iv*) Suppose $P = \overline{x_i}\langle x_j \rangle.P_1$

$$\begin{aligned}
&\left[\!\left[N(s + 1_a + 1_b + t) \vdash \overline{x_i}\langle x_j \rangle.P_1 \right]\!\right] \\
= \;& out \circ \langle \left[\!\left[N(s + 1_a + 1_b + t) \vdash x_i \right]\!\right], \left[\!\left[N(s + 1_a + 1_b + t) \vdash x_j \right]\!\right], \\
& \left[\!\left[N(s + 1_a + 1_b + t) \vdash P_1 \right]\!\right] \rangle \\
= \;& out \circ \langle \left[\!\left[N(s + 1_b + 1_a + t) \vdash x_i \right]\!\right] \circ \pi_P, \left[\!\left[N(s + 1_b + 1_a + t) \vdash x_j \right]\!\right] \circ \pi_P, \\
& \left[\!\left[N(s + 1_b + 1_a + t) \vdash P_1 \right]\!\right] \circ \pi_P \rangle \\
= \;& out \circ \langle \left[\!\left[N(s + 1_b + 1_a + t) \vdash x_i \right]\!\right], \left[\!\left[N(s + 1_b + 1_a + t) \vdash x_j \right]\!\right], \\
& \left[\!\left[N(s + 1_b + 1_a + t) \vdash P_1 \right]\!\right] \rangle \circ \pi_P \\
= \;& \left[\!\left[N(s + 1_b + 1_a + t) \vdash \overline{x_i}\langle x_j \rangle.P_1 \right]\!\right] \circ \pi_P
\end{aligned}$$

(v) Suppose $P = \overline{x_i}(z).P_1$

$$\begin{aligned}
& [\![N(s + 1_a + 1_b + t) \vdash \overline{x_i}(z).P_1]\!] \\
= \; & \mathit{bout} \circ \langle [\![N(s + 1_a + 1_b + t) \vdash x_i]\!], \Lambda [\![N(s + 1_a + 1_b + t + 1) \vdash P_1]\!] \rangle \\
= \; & \mathit{bout} \circ \langle [\![N(s + 1_b + 1_a + t) \vdash x_i]\!] \circ \pi_P, \\
& \Lambda([\![N(s + 1_b + 1_a + t + 1) \vdash P_1]\!] \circ \pi_P) \rangle \\
= \; & \mathit{bout} \circ \langle [\![N(s + 1_b + 1_a + t) \vdash x_i]\!] \circ \pi_P, \\
& \Lambda([\![N(s + 1_b + 1_a + t + 1) \vdash P_1]\!]) \circ \pi_P \rangle \\
= \; & \mathit{bout} \circ \langle [\![N(s + 1_b + 1_a + t) \vdash x_i]\!], \\
& \Lambda [\![N(s + 1_b + 1_a + t + 1) \vdash P_1]\!] \rangle \circ \pi_P \\
= \; & [\![N(s + 1_b + 1_a + t) \vdash \overline{x_i}(z).P_1]\!] \circ \pi_P
\end{aligned}$$

(vi) Suppose $P = \nu z P_1$

$$\begin{aligned}
& [\![N(s + 1_a + 1_b + t) \vdash \nu z P_1]\!] \\
= \; & \mathit{res} \circ \langle \Lambda [\![N(s + 1_a + 1_b + t + 1) \vdash P_1]\!] \rangle \\
= \; & \mathit{res} \circ \langle \Lambda([\![N(s + 1_b + 1_a + t + 1) \vdash P_1]\!] \circ \pi_P) \rangle \\
= \; & \mathit{res} \circ \langle \Lambda([\![N(s + 1_b + 1_a + t + 1) \vdash P_1]\!]) \circ \pi_P \rangle \\
= \; & \mathit{res} \circ \langle \Lambda [\![N(s + 1_b + 1_a + t + 1) \vdash P_1]\!] \rangle \circ \pi_P \\
= \; & [\![N(s + 1_b + 1_a + t) \vdash \nu z P_1]\!] \circ \pi_P
\end{aligned}$$

(vii) Suppose $P = P_1 | P_2$

$$\begin{aligned}
& [\![N(s + 1_a + 1_b + t) \vdash P_1 | P_2]\!] \\
= \; & \mathit{par} \circ \langle [\![N(s + 1_a + 1_b + t) \vdash P_1]\!], [\![N(s + 1_a + 1_b + t) \vdash P_2]\!] \rangle \\
= \; & \mathit{par} \circ \langle [\![N(s + 1_b + 1_a + t) \vdash P_1]\!] \circ \pi_P, [\![N(s + 1_b + 1_a + t) \vdash P_2]\!] \circ \pi_P \rangle \\
= \; & \mathit{par} \circ \langle [\![N(s + 1_b + 1_a + t) \vdash P_1]\!], [\![N(s + 1_b + 1_a + t) \vdash P_2]\!] \rangle \circ \pi_P \\
= \; & [\![N(s + 1_b + 1_a + t) \vdash P_1 | P_2]\!] \circ \pi_P
\end{aligned}$$

($viii$) Suppose $P = P_1 + P_2$, $P = P_1 || P_2$, or $P = P_1 \downarrow P_2$, the result is shown similarly as in (vii).

2. (CONTRACTION) Suppose $N(s + t) \vdash P$ and $a' \in N(1_a)$, we will show by structural induction on P that:

$$[\![N(s + 1_a + t) \vdash P]\!] = [\![N(s + t) \vdash P]\!] \circ \pi_C$$

Let,

- $N(s+1_a+t) = \{x_1, x_2, \ldots, x_{|N(s)|}, a' = x_{|N(s)|+1}, x_{|N(s)|+2}, \ldots, x_{|N(s)|+|N(t)|+1}\}$,
- $z \in N(1)$, and
- **Induction Hypothesis:**
  $[\![N(s+1_a+t \vdash P_i]\!] = [\![N(s+t) \vdash P_i]\!] \circ \pi_C$ for $i = 1, 2$.

(i) Suppose $N(s+t) \vdash x_i$ (i.e. $x_i \neq a'$). Let $s' \in |\mathbf{Inj}|$, and $y_j \in N(s'), 1 \leq j \leq |N(s)| + |N(t)| + 1$:

$$\begin{aligned}
\text{LHS} &= [\![N(s+1_a+t) \vdash x_i]\!]_{s'}(y_1, \ldots, y_{|N(s)|+|N(t)|+1}) \\
&= \pi_i(y_1, \ldots, y_{|N(s)|+|N(t)|+1}) \\
&= y_i \\
\text{RHS} &= [\![N(s+t) \vdash x_i]\!]_{s'} \circ \pi_C(y_1, \ldots, y_{|N(s)|+|N(t)|+1}) \\
&= \begin{cases} \pi_i(y_1, \ldots, y_{|N(s)|}, y_{|N(s)|+2}, \\ y_{|N(s)|+3}, \ldots, y_{|N(s)|+|N(t)|+1}) & \text{if } i \leq |N(s)| \\ \pi_{i-1}(y_1, \ldots, y_{|N(s)|}, y_{|N(s)|+2}, \\ y_{|N(s)|+3}, \ldots, y_{|N(s)|+|N(t)|+1}) & \text{if } i \geq |N(s)|+2 \end{cases} \\
&= y_i
\end{aligned}$$

We have that RHS = LHS.

(ii) Suppose $P = 0$. Let $s' \in |\mathbf{Inj}|$, and $y_j \in N(s')$, for $1 \leq j \leq |N(s)|+|N(t)|+1$:

$$\begin{aligned}
\text{LHS} &= [\![N(s+1_a+t) \vdash 0]\!]_{s'}(y_1, \ldots, y_{|N(s)|+|N(t)|+1}) \\
&= nil_{s+1_a+t} \\
\text{RHS} &= [\![N(s+t) \vdash 0]\!]_{s'} \circ \pi_C(y_1, \ldots, y_{|N(s)|+|N(t)|+1}) \\
&= [\![N(s+t) \vdash 0]\!]_{s'}(y_1, \ldots, y_{|N(s)|}, y_{|N(s)|+2}, y_{|N(s)|+3}, \ldots, y_{|N(s)|+|N(t)|+1}) \\
&= nil_{s+t}
\end{aligned}$$

We have that RHS = LHS.

(*iii*) Suppose $P = x_i(z).P_1$

$$\begin{aligned}
&\llbracket N(s+1_a+t) \vdash x_i(z).P_1 \rrbracket \\
= \;& in \circ \langle \llbracket N(s+1_a+t) \vdash x_i \rrbracket, \lambda \llbracket N(s+1_a+t+1) \vdash P_1 \rrbracket \rangle \\
= \;& in \circ \langle \llbracket N(s+t) \vdash x_i \rrbracket \circ \pi_C, \lambda(\llbracket N(s+t+1) \vdash P_1 \rrbracket \circ \pi_C) \rangle \\
= \;& in \circ \langle \llbracket N(s+t) \vdash x_i \rrbracket \circ \pi_C, \lambda(\llbracket N(s+t+1) \vdash P_1 \rrbracket) \circ \pi_C \rangle \\
= \;& in \circ \langle \llbracket N(s+t) \vdash x_i \rrbracket, \lambda \llbracket N(s+t+1) \vdash P_1 \rrbracket \rangle \circ \pi_C \\
= \;& \llbracket N(s+t) \vdash x_i(z).P_1 \rrbracket \circ \pi_C
\end{aligned}$$

(*iv*) Suppose $P = \overline{x_i}\langle x_j \rangle.P_1$

$$\begin{aligned}
&\llbracket N(s+1_a+t) \vdash \overline{x_i}\langle x_j \rangle.P_1 \rrbracket \\
= \;& out \circ \langle \llbracket N(s+1_a+t) \vdash x_i \rrbracket, \llbracket N(s+1_a+t) \vdash x_j \rrbracket, \llbracket N(s+1_a+t) \vdash P_1 \rrbracket \rangle \\
= \;& out \circ \langle \llbracket N(s+t) \vdash x_i \rrbracket \circ \pi_C, \llbracket N(s+t) \vdash x_j \rrbracket \circ \pi_C, \llbracket N(s+t) \vdash P_1 \rrbracket \circ \pi_C \rangle \\
= \;& out \circ \langle \llbracket N(s+t) \vdash x_i \rrbracket, \llbracket N(s+t) \vdash x_j \rrbracket, \llbracket N(s+t) \vdash P_1 \rrbracket \rangle \circ \pi_C \\
= \;& \llbracket N(s+t) \vdash \overline{x_i}\langle x_j \rangle.P_1 \rrbracket \circ \pi_C
\end{aligned}$$

(*v*) Suppose $P = \overline{x_i}(z).P_1$

$$\begin{aligned}
&\llbracket N(s+1_a+t) \vdash \overline{x_i}(z).P_1 \rrbracket \\
= \;& bout \circ \langle \llbracket N(s+1_a+t) \vdash x_i \rrbracket, \Lambda \llbracket N(s+1_a+t+1) \vdash P_1 \rrbracket \rangle \\
= \;& bout \circ \langle \llbracket N(s+t) \vdash x_i \rrbracket \circ \pi_C, \Lambda(\llbracket N(s+t+1) \vdash P_1 \rrbracket \circ \pi_C) \rangle \\
= \;& bout \circ \langle \llbracket N(s+t) \vdash x_i \rrbracket \circ \pi_C, \Lambda(\llbracket N(s+t+1) \vdash P_1 \rrbracket) \circ \pi_C \rangle \\
= \;& bout \circ \langle \llbracket N(s+t) \vdash x_i \rrbracket, \Lambda \llbracket N(s+t+1) \vdash P_1 \rrbracket \rangle \circ \pi_C \\
= \;& \llbracket N(s+t) \vdash \overline{x_i}(z).P_1 \rrbracket \circ \pi_C
\end{aligned}$$

(*vi*) Suppose $P = \nu z P_1$

$$\begin{aligned}
&\llbracket N(s+1_a+t) \vdash \nu z P_1 \rrbracket \\
= \;& res \circ \langle \Lambda \llbracket N(s+1_a+t+1) \vdash P_1 \rrbracket \rangle \\
= \;& res \circ \langle \Lambda(\llbracket N(s+t+1) \vdash P_1 \rrbracket \circ \pi_C) \rangle \\
= \;& res \circ \langle \Lambda(\llbracket N(s+t+1) \vdash P_1 \rrbracket) \circ \pi_C \rangle \\
= \;& res \circ \langle \Lambda \llbracket N(s+t+1) \vdash P_1 \rrbracket \rangle \circ \pi_C \\
= \;& \llbracket N(s+t) \vdash \nu z P_1 \rrbracket \circ \pi_C
\end{aligned}$$

(*vii*) Suppose $P = P_1|P_2$

$$\begin{aligned}
& \left[\!\left[N(s+1_a+t) \vdash P_1|P_2 \right]\!\right] \\
= \; & par \circ \langle \left[\!\left[N(s+1_a+t) \vdash P_1 \right]\!\right], \left[\!\left[N(s+1_a+t) \vdash P_2 \right]\!\right] \rangle \\
= \; & par \circ \langle \left[\!\left[N(s+t) \vdash P_1 \right]\!\right] \circ \pi_C, \left[\!\left[N(s+t) \vdash P_2 \right]\!\right] \circ \pi_C \rangle \\
= \; & par \circ \langle \left[\!\left[N(s+t) \vdash P_1 \right]\!\right], \left[\!\left[N(s+t) \vdash P_2 \right]\!\right] \rangle \circ \pi_C \\
= \; & \left[\!\left[N(s+t) \vdash P_1|P_2 \right]\!\right] \circ \pi_C
\end{aligned}$$

(*viii*) Suppose $P = P_1 + P_2$, $P = P_1||P_2$, or $P = P_1 \downarrow P_2$, the result is shown similarly as in (*vii*).

3. (SUBSTITUTION) Suppose $N(s+1_a+t) \vdash P$, $a' \in N(1_a)$, and $x_j \in N(s+t)$, we will show by structural induction on P that

$$\left[\!\left[N(s+t) \vdash P[a' := x_j] \right]\!\right] = \left[\!\left[N(s+1_a+t) \vdash P \right]\!\right] \circ \pi_S$$

Let,

- $N(s+1_a+t) = \{x_1, x_2, \ldots, x_{|N(s)|}, a' = x_{|N(s)|+1}, x_{|N(s)|+2}, \ldots, x_{|N(s)|+|N(t)|+1}\}$,
- $z \in N(1)$, and
- **Induction Hypothesis:**
  $\left[\!\left[N(s+t) \vdash P_i[a' := x_j] \right]\!\right] = \left[\!\left[N(s+1_a+t) \vdash P_i \right]\!\right] \circ \pi_S$ for $i = 1, 2$.

(*i*) There are 2 cases to check for the interpretation of free names. Let $s' \in |\mathbf{Inj}|$, and $y_j \in N(s'), 1 \leq j \leq |N(s)| + |N(t)|$:

(a) Suppose the free variable is $a' = x_{|N(s)|+1}$

$$\begin{aligned}
\text{LHS} &= [\![N(s+t) \vdash a'[a' := x_j]]\!]_{s'}(y_1, \ldots, y_{|N(s)|+|N(t)|}) \\
&= [\![N(s+t) \vdash x_j]\!]_{s'}(y_1, \ldots, y_{|N(s)|+|N(t)|}) \\
&= \begin{cases} \pi_j(y_1, \ldots, y_{|N(s)|+|N(t)|}) & \text{if } j \leq |N(s)| \\ \pi_{j-1}(y_1, \ldots, y_{|N(s)|+|N(t)|}) & \text{if } j \geq |N(s)|+2 \end{cases} \\
&= \begin{cases} y_j & \text{if } j \leq |N(s)| \\ y_{j-1} & \text{if } j \geq |N(s)|+2 \end{cases} \\
\text{RHS} &= [\![N(s+1_a+t) \vdash a']\!]_{s'} \circ \pi_S(y_1, \ldots, y_{|N(s)|+|N(t)|}) \\
&= \begin{cases} \pi_{|N(s)|+1}(y_1, \ldots, y_{|N(s)|}, & \\ y_j, y_{|N(s)|+1}, \cdots y_{|N(s)|+|N(t)|}) & \text{if } j \leq |N(s)| \\ \pi_{|N(s)|+1}(y_1, \ldots, y_{|N(s)|}, & \\ y_{j-1}, y_{|N(s)|+1}, \cdots y_{|N(s)|+|N(t)|}) & \text{if } j \geq |N(s)|+2 \end{cases} \\
&= \begin{cases} y_j & \text{if } j \leq |N(s)| \\ y_{j-1} & \text{if } j \geq |N(s)|+2 \end{cases}
\end{aligned}$$

We have that RHS = LHS.

(b) Suppose the free variable is $x_i \in N(s+t)$, $x_i \neq a'$.

$$\begin{aligned}
\text{LHS} &= [\![N(s+t) \vdash x_i[a' := x_j]]\!]_{s'}(y_1, \ldots, y_{|N(s)|+|N(t)|}) \\
&= [\![N(s+t) \vdash x_i]\!]_{s'}(y_1, \ldots, y_{|N(s)|+|N(t)|}) \\
&= \begin{cases} \pi_i(y_1, \ldots, y_{|N(s)|+|N(t)|}) & \text{if } i \leq |N(s)| \\ \pi_{i-1}(y_1, \ldots, y_{|N(s)|+|N(t)|}) & \text{if } i \geq |N(s)|+2 \end{cases} \\
&= \begin{cases} y_i & \text{if } i \leq |N(s)| \\ y_{i-1} & \text{if } i \geq |N(s)|+2 \end{cases} \\
\text{RHS} &= [\![N(s+1_a+t) \vdash x_i]\!]_{s'} \circ \pi_S(y_1, \ldots, y_{|N(s)|+|N(t)|+1}) \\
&= \pi_i(y_1, \ldots, y_{|N(s)|}, y_j, y_{|N(s)|+1}, \cdots y_{|N(s)|+|N(t)|}) \\
&= \begin{cases} y_i & \text{if } i \leq |N(s)| \\ y_{i-1} & \text{if } i \geq |N(s)|+2 \end{cases}
\end{aligned}$$

We have that RHS = LHS.

(*ii*) Suppose $P = 0$. Let $s' \in |\mathbf{Inj}|$, and $y_j \in N(s'), 1 \leq j \leq |N(s)| + |N(t)|$.

$$\begin{aligned}
\text{LHS} &= \left[\!\left[N(s+t) \vdash 0[a' := x_j] \right]\!\right]_{s'}(y_1, \ldots, j_{|N(s)|+|N(t)|}) \\
&= \left[\!\left[N(s+t) \vdash 0 \right]\!\right]_{s'}(y_1, \ldots, j_{|N(s)|+|N(t)|}) \\
&= nil_{s+t} \\
\text{RHS} &= \left[\!\left[N(s+1_a+t) \vdash 0 \right]\!\right]_{s'} \circ \pi_S(y_1, \ldots, y_{|N(s)|+|N(t)|}) \\
&= \left[\!\left[N(s+1_a+t) \vdash 0 \right]\!\right]_{s'}(y_1, \ldots, y_{|N(s)|}, y_j, y_{|N(s)|+1}, \ldots, y_{|N(s)|+|N(t)|}) \\
&= nil_{s+1_a+t}
\end{aligned}$$

We have that RHS = LHS.

(*iii*) Suppose $P = x_i(z).P_1$

$$\begin{aligned}
& \left[\!\left[N(s+t) \vdash x_i(z).P_1[a' := x_j] \right]\!\right] \\
= \; & in \circ \langle \left[\!\left[N(s+t) \vdash x_i[a' := x_j] \right]\!\right], \lambda \left[\!\left[N(s+t+1) \vdash P_1[a' := x_j] \right]\!\right] \rangle \\
= \; & in \circ \langle \left[\!\left[N(s+1_a+t) \vdash x_i \right]\!\right] \circ \pi_S, \lambda(\left[\!\left[N(s+1_a+t+1) \vdash P_1 \right]\!\right] \circ \pi_S) \rangle \\
= \; & in \circ \langle \left[\!\left[N(s+1_a+t) \vdash x_i \right]\!\right] \circ \pi_S, \lambda(\left[\!\left[N(s+1_a+t+1) \vdash P_1 \right]\!\right]) \circ \pi_S \rangle \\
= \; & in \circ \langle \left[\!\left[N(s+1_a+t) \vdash x_i \right]\!\right], \lambda \left[\!\left[N(s+1_a+t+1) \vdash P_1 \right]\!\right] \rangle \circ \pi_S \\
= \; & \left[\!\left[N(s+1_a+t) \vdash x_i(z).P_1 \right]\!\right] \circ \pi_S
\end{aligned}$$

(*iv*) Suppose $P = \overline{x_i}\langle x_j \rangle.P_1$

$$\begin{aligned}
& \left[\!\left[N(s+t) \vdash \overline{x_i}\langle x_j \rangle.P_1[a' := x_j] \right]\!\right] \\
= \; & out \circ \langle \left[\!\left[N(s+t) \vdash x_i[a' := x_j] \right]\!\right], \left[\!\left[N(s+t) \vdash x_j[a' := x_j] \right]\!\right], \\
& \left[\!\left[N(s+t) \vdash P_1[a' := x_j] \right]\!\right] \rangle \\
= \; & out \circ \langle \left[\!\left[N(s+1_a+t) \vdash x_i \right]\!\right] \circ \pi_S, \left[\!\left[N(s+1_a+t) \vdash x_j \right]\!\right] \circ \pi_S, \\
& \left[\!\left[N(s+1_a+t) \vdash P_1 \right]\!\right] \circ \pi_S \rangle \\
= \; & out \circ \langle \left[\!\left[N(s+1_a+t) \vdash x_i \right]\!\right], \left[\!\left[N(s+1_a+t) \vdash x_j \right]\!\right], \\
& \left[\!\left[N(s+1_a+t) \vdash P_1 \right]\!\right] \rangle \circ \pi_S \\
= \; & \left[\!\left[N(s+t) \vdash \overline{x_i}\langle x_j \rangle.P_1 \right]\!\right] \circ \pi_S
\end{aligned}$$

(v) Suppose $P = \overline{x_i}(z).P_1$

$$\begin{aligned}
& [\![N(s+t) \vdash \overline{x_i}(z).P_1[a' := x_j]]\!] \\
= \; & \mathit{bout} \circ \langle [\![N(s+t) \vdash x_i[a' := x_j]]\!], \Lambda [\![N(s+t+1) \vdash P_1[a' := x_j]]\!] \rangle \\
= \; & \mathit{bout} \circ \langle [\![N(s+1_a+t) \vdash x_i]\!] \circ \pi_S, \Lambda([\![N(s+1_a+t+1) \vdash P_1]\!] \circ \pi_S) \rangle \\
= \; & \mathit{bout} \circ \langle [\![N(s+1_a+t) \vdash x_i]\!] \circ \pi_S, \Lambda([\![N(s+1_a+t+1) \vdash P_1]\!]) \circ \pi_S \rangle \\
= \; & \mathit{bout} \circ \langle [\![N(s+1_a+t) \vdash x_i]\!], \Lambda [\![N(s+1_a+t+1) \vdash P_1]\!] \rangle \circ \pi_S \\
= \; & [\![N(s+1_a+t) \vdash \overline{x_i}(z).P_1]\!] \circ \pi_S
\end{aligned}$$

(vi) Suppose $P = \nu z P_1$

$$\begin{aligned}
& [\![N(s+t) \vdash \nu z P_1[a' := x_j]]\!] \\
= \; & \mathit{res} \circ \langle \Lambda [\![N(s+t+1) \vdash P_1[a' := x_j]]\!] \rangle \\
= \; & \mathit{res} \circ \langle \Lambda([\![N(s+1_a+t+1) \vdash P_1]\!] \circ \pi_S) \rangle \\
= \; & \mathit{res} \circ \langle \Lambda([\![N(s+1_a+t+1) \vdash P_1]\!]) \circ \pi_S \rangle \\
= \; & \mathit{res} \circ \langle \Lambda [\![N(s+1_a+t+1) \vdash P_1]\!] \rangle \circ \pi_S \\
= \; & [\![N(s+1_a+t) \vdash \nu z P_1]\!] \circ \pi_S
\end{aligned}$$

(vii) Suppose $P = P_1 | P_2$

$$\begin{aligned}
& [\![N(s+t) \vdash (P_1|P_2)[a' := x_j]]\!] \\
= \; & [\![N(s+t) \vdash P_1[a' := x_j] | P_2[a' := x_j]]\!] \\
= \; & \mathit{par}_s \circ \langle [\![N(s+t) \vdash P_1[a' := x_j]]\!], [\![N(s+t) \vdash P_2[a' := x_j]]\!] \rangle \\
= \; & \mathit{par} \circ \langle [\![N(s+1_a+t) \vdash P_1]\!] \circ \pi_S, [\![N(s+1_a+t) \vdash P_2]\!] \circ \pi_S \rangle \\
= \; & \mathit{par} \circ \langle [\![N(s+1_a+t) \vdash P_1]\!], [\![N(s+1_a+t) \vdash P_2]\!] \rangle \circ \pi_S \\
= \; & [\![N(s+1_a+t) \vdash P_1|P_2]\!] \circ \pi_S
\end{aligned}$$

($viii$) Suppose $P = P_1 + P_2$, $P = P_1 || P_2$, or $P = P_1 \downarrow P_2$, the result is shown similarly as in (vii).

4. (ALPHA-CONVERSION) For $z \in N(1)$, and $z' \in N(1_z)$, there are three cases to check:

$$(i)\ \left[\!\left[N(s) \vdash x_i(z).P \right]\!\right] = \left[\!\left[N(s) \vdash x_i(z').(P[z := z']) \right]\!\right]$$

$$\begin{aligned}
&\left[\!\left[N(s) \vdash x_i(z').(P[z := z']) \right]\!\right] \\
&= in \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \lambda \left[\!\left[N(s+1_z) \vdash P[z := z'] \right]\!\right] \rangle \\
&= in \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \lambda (\left[\!\left[N(s+1+1_z \vdash P \right]\!\right] \circ \pi_S) \rangle \\
&= in \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \lambda (\left[\!\left[N(s+1+1_z) \vdash P \right]\!\right]) \circ \pi_S \rangle \\
&= in \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \lambda (\left[\!\left[N(s+1) \vdash P \right]\!\right] \circ \pi_C) \circ \pi_S \rangle \\
&= in \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \lambda (\left[\!\left[N(s+1) \vdash P \right]\!\right]) \circ \pi_C \circ \pi_S \rangle \\
&= in \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \lambda \left[\!\left[N(s+1) \vdash P \right]\!\right] \rangle \\
&= \left[\!\left[N(s) \vdash x_i(z).P \right]\!\right]
\end{aligned}$$

$$(ii)\ \left[\!\left[N(s) \vdash \nu z P \right]\!\right] = \left[\!\left[N(s) \vdash \nu z'(P[z := z'] \right]\!\right]$$

$$\begin{aligned}
&\left[\!\left[N(s) \vdash \nu z'(P[z := z']) \right]\!\right] \\
&= res \circ \langle \Lambda \left[\!\left[N(s+1_z \vdash P[z := z'] \right]\!\right] \rangle \\
&= res \circ \langle \Lambda (\left[\!\left[N(s+1+1_z) \vdash P \right]\!\right] \circ \pi_S) \rangle \\
&= res \circ \langle \Lambda (\left[\!\left[N(s+1+1_z) \vdash P \right]\!\right]) \circ \pi_S \rangle \\
&= res \circ \langle \Lambda (\left[\!\left[N(s+1) \vdash P \right]\!\right] \circ \pi_C) \circ \pi_S \rangle \\
&= res \circ \langle \Lambda (\left[\!\left[N(s+1) \vdash P \right]\!\right]) \circ \pi_C \circ \pi_S \rangle \\
&= res \circ \langle \Lambda \left[\!\left[N(s+1) \vdash P \right]\!\right] \rangle \\
&= \left[\!\left[N(s) \vdash \nu z P \right]\!\right]
\end{aligned}$$

$$(iii)\ \left[\!\left[N(s) \vdash \overline{x_i}(z).P \right]\!\right] = \left[\!\left[N(s) \vdash \overline{x_i}(z').(P[z := z']) \right]\!\right]$$

$$\begin{aligned}
&\left[\!\left[N(s) \vdash \overline{x_i}(z').(P[z := z']) \right]\!\right] \\
&= bout \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \Lambda \left[\!\left[N(s+1_z) \vdash P[z := z'] \right]\!\right] \rangle \\
&= bout \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \Lambda (\left[\!\left[N(s+1+1_z \vdash P \right]\!\right] \circ \pi_S) \rangle \\
&= bout \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \Lambda (\left[\!\left[N(s+1+1_z \vdash P \right]\!\right]) \circ \pi_S \rangle \\
&= bout \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \Lambda (\left[\!\left[N(s+1) \vdash P \right]\!\right] \circ \pi_C) \circ \pi_S \rangle \\
&= bout \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \Lambda (\left[\!\left[N(s+1) \vdash P \right]\!\right]) \circ \pi_C \circ \pi_S \rangle \\
&= bout \circ \langle \left[\!\left[N(s) \vdash x_i \right]\!\right], \Lambda \left[\!\left[N(s+1) \vdash P \right]\!\right] \rangle \\
&= \left[\!\left[N(s) \vdash \overline{x_i}(z).P \right]\!\right]
\end{aligned}$$

□

Appendix B

Soundness: Proof of Theorem 4.1.10

We prove the following result:

Theorem B.1 *Let P and Q be two Finite π-calculus processes such that $N(s) \vdash P$ and $N(s) \vdash Q$, where $N(s) = \{x_1, x_2, \ldots, x_n\}$. If $SGE \vdash P = Q$, that is the processes are late-bisimular, then $(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash Q|\!)$.*

Proof. We proceed by proving that the closed interpretation preserves the SGE axioms and inference rules which generate all the late-bisimular processes in the Finite π-calculus.

1. Alpha-Conversion

AC1 Suppose $P \equiv_\alpha Q$, then we must prove that $(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash Q|\!)$. Hence we must verify three cases: Let $z \in N(1)$, and $z' \in N(1_z)$,

(i) $P \equiv_\alpha Q$. where $P = x_i(z).P'$ and $Q = x_i(z').(P'[z := z'])$

$$
\begin{aligned}
(\!|N(s) \vdash x_i(z).P'|\!) &= in_s\langle (\!|N(s) \vdash x_i|\!), \langle (\!|N(s) \vdash P'[z = x_j]|\!)\rangle_{j=1}^n, (\!|N(s+1) \vdash P'|\!)\rangle \\
\text{by Cor 4.1.7} &= in_s\langle (\!|N(s) \vdash x_i|\!), \langle [\![N(s) \vdash P'[z := x_j]]\!]_s(x_1, x_2, \ldots, x_n)\rangle_{j=1}^n, \\
&\quad (\!|N(s+1_z) \vdash P'[z := z']|\!)\rangle \\
\text{by Lemma 4.1.6} &= in_s\langle (\!|N(s) \vdash x_i|\!), \langle ([\![N(s+1) \vdash P']\!] \circ \pi_S)_s(x_1, x_2, \ldots, x_n)\rangle_{j=1}^n, \\
&\quad (\!|N(s+1_z) \vdash P'[z := z']|\!)\rangle \\
\text{by Cor 4.1.7} &= in_s\langle (\!|N(s) \vdash x_i|\!), \\
&\quad \langle ([\![N(s+1_z) \vdash P'[z := z']]\!] \circ \pi_S)_s(x_1, x_2, \ldots, x_n)\rangle_{j=1}^n, \\
&\quad (\!|N(s+1_z) \vdash P'[z := z']|\!)\rangle \\
\text{by Lemma 4.1.6} &= in_s\langle (\!|N(s) \vdash x_i|\!), \\
&\quad \langle [\![N(s) \vdash (P'[z := z'])[z' := x_j]]\!]_s(x_1, x_2, \ldots, x_n)\rangle_{j=1}^n, \\
&\quad (\!|N(s+1_z) \vdash P'[z := z']|\!)\rangle \\
&= in_s\langle (\!|N(s) \vdash x_i|\!), \langle (\!|N(s) \vdash (P'[z := z'])[z' := x_j]|\!)\rangle_{j=1}^n, \\
&\quad (\!|N(s+1_z) \vdash P[z := z']|\!)\rangle \\
&= (\!|N(s) \vdash x_i(z').(P[z := z'])|\!)
\end{aligned}
$$

(ii) $P \equiv_\alpha Q$, where $P = \overline{x_i}(z).P'$ and $Q = \overline{x_i}(z').(P'[z := z'])$

$$
\begin{aligned}
(\!|N(s) \vdash x_i(z).P'|\!) &= bout_s\langle (\!|N(s) \vdash x_i|\!), (\!|N(s+1) \vdash P'|\!)\rangle \\
\text{by Cor 4.1.7} &= bout_s\langle (\!|N(s) \vdash x_i|\!), (\!|N(s+1_z) \vdash P[z := z']|\!)\rangle \\
&= (\!|N(s) \vdash \overline{x_i}(z').(P[z := z'])|\!)
\end{aligned}
$$

(iii) $P \equiv_\alpha Q$, where $P = \nu z P'$ and $Q = \nu z'(P'[z := z'])$

$$
\begin{aligned}
(\!|N(s) \vdash \nu z P'|\!) &= res_s\langle (\!|N(s+1) \vdash P'|\!)\rangle \\
\text{by Cor 4.1.7} &= res_s\langle (\!|N(s+1_z) \vdash P[z := z']|\!)\rangle \\
&= (\!|N(s) \vdash \nu z'(P[z := z'])|\!)
\end{aligned}
$$

2. Summation

S1 $SGE \vdash P + 0 = P$

$$
\begin{aligned}
(\!|N(s) \vdash P + 0|\!) &= sum_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash 0|\!)\rangle \\
&= sum_s\langle (\!|N(s) \vdash P|\!), nil_s\rangle \\
\text{by Property 8(b),S1} &= (\!|N(s) \vdash P|\!)
\end{aligned}
$$

S2 $SGE \vdash P + Q = Q + P$

$$\begin{aligned} (\!|N(s) \vdash P + Q|\!) &= sum_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q|\!)\rangle \\ \text{by Property 8(b),}\mathbf{S2} &= sum_s\langle (\!|N(s) \vdash Q|\!), (\!|N(s) \vdash P|\!)\rangle \\ &= (\!|N(s) \vdash Q + P|\!) \end{aligned}$$

S3 $SGE \vdash P + (Q + R) = (P + Q) + R$

$$\begin{aligned} (\!|N(s) \vdash P + (Q + R)|\!) &= sum_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q + R|\!)\rangle \\ &= sum_s\langle (\!|N(s) \vdash P|\!), sum_s\langle (\!|N(s) \vdash Q|\!), (\!|N(s) \vdash R|\!)\rangle\rangle \\ \text{by Property 8(b),}\mathbf{S3} &= sum_s\langle sum_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q|\!)\rangle, (\!|N(s) \vdash R|\!)\rangle \\ &= sum_s\langle (\!|N(s) \vdash P + Q|\!), (\!|N(s) \vdash R|\!)\rangle \\ &= (\!|N(s) \vdash (P + Q) + R|\!) \end{aligned}$$

S4 $SGE \vdash P + P = P$

$$\begin{aligned} (\!|N(s) \vdash P + P|\!) &= sum_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash P|\!)\rangle \\ \text{by Property 8(b),}\mathbf{S4} &= (\!|N(s) \vdash P|\!) \end{aligned}$$

3. Restriction

R1 $SGE \vdash \nu z(P + Q) = \nu zP + \nu zQ$

$$\begin{aligned} (\!|N(s) \vdash \nu z(P + Q)|\!) &= res_s\langle (\!|N(s+1) \vdash P + Q|\!)\rangle \\ &= res_s\langle sum_{s+1}\langle (\!|N(s+1) \vdash P|\!), (\!|N(s+1) \vdash Q|\!)\rangle\rangle \\ \text{by Property 8(a),}\mathbf{R5} &= sum_s\langle res_s\langle (\!|N(s+1) \vdash P|\!)\rangle, res_s\langle (\!|N(s+1) \vdash Q|\!)\rangle\rangle \\ &= sum_s\langle (\!|N(s) \vdash \nu zP|\!), (\!|N(s) \vdash \nu zQ|\!)\rangle \\ &= (\!|N(s) \vdash \nu zP + \nu zQ|\!) \end{aligned}$$

R2 if $z \notin names(\alpha)$ then $SGE \vdash \nu z\alpha.P = \alpha.\nu zP$. There are a four cases to check:

(i) Suppose $\alpha = x_i(z')$

$$\begin{aligned}
(\!|N(s) \vdash \nu z\alpha.P|\!) &= res_s\langle(\!|N(s+1) \vdash \alpha.P|\!)\rangle \\
&= res_s\langle in_{s+1}\langle(\!|N(s+1) \vdash x_i|\!), \langle(\!|N(s+1) \vdash P[z' := x_j]|\!)\rangle_{j=1}^n, \\
&\quad (\!|N(s+1) \vdash P[z' := z]|\!), (\!|N((s+1)+1_z) \vdash P|\!)\rangle\rangle \\
\text{by Property 8(a),}\mathbf{R1} &= in_s\langle(\!|N(s) \vdash x_i|\!), \langle res_s\langle(\!|N(s+1) \vdash P[z' := x_j]|\!)\rangle\rangle_{j=1}^n, \\
&\quad res_s\langle(\!|N((s+1)+1_z) \vdash P|\!)\rangle\rangle \\
&= in_s\langle(\!|N(s) \vdash x_i|\!), \langle(\!|N(s) \vdash \nu z P[z' := x_j]|\!)\rangle_{j=1}^n, \\
&\quad (\!|N(s+1_z) \vdash \nu z P|\!)\rangle \\
&= (\!|N(s) \vdash \alpha.\nu z P|\!)
\end{aligned}$$

(ii) Suppose $\alpha = \overline{x_i}\langle x_j\rangle$

$$\begin{aligned}
(\!|N(s) \vdash \nu z\alpha.P|\!) &= res_s\langle(\!|N(s+1) \vdash \alpha.P|\!)\rangle \\
&= res_s\langle out_{s+1}\langle(\!|N(s+1) \vdash x_i|\!), (\!|N(s+1) \vdash x_j|\!), \\
&\quad (\!|N(s+1) \vdash P|\!)\rangle\rangle \\
\text{by Property 8(a),}\mathbf{R2} &= out_s\langle(\!|N(s) \vdash x_i|\!), (\!|N(s) \vdash x_j|\!), res_s\langle(\!|N(s+1) \vdash P|\!)\rangle\rangle \\
&= out_s\langle(\!|N(s) \vdash x_i|\!), (\!|N(s) \vdash x_j|\!), (\!|N(s) \vdash \nu z P|\!)\rangle \\
&= (\!|N(s) \vdash \alpha.\nu z P|\!)
\end{aligned}$$

(iii) Suppose $\alpha = \overline{x_i}(z')$

$$\begin{aligned}
(\!|N(s) \vdash \nu z\alpha.P|\!) &= res_s\langle(\!|N(s+1) \vdash \alpha.P|\!)\rangle \\
&= res_s\langle bout_{s+1}\langle(\!|N(s+1) \vdash x_i|\!), (\!|N((s+1)+1_z) \vdash P|\!)\rangle\rangle \\
\text{by Property 8(a),}\mathbf{R3} &= bout_s\langle(\!|N(s) \vdash x_i|\!), res_s\langle(\!|N((s+1)+1_z) \vdash P|\!)\rangle\rangle \\
\text{by Lemma 4.1.6,4.1.9} &= bout_s\langle(\!|N(s) \vdash x_i|\!), res_s\langle(\!|N((s+1_z)+1) \vdash P|\!)\rangle\rangle \\
&= bout_s\langle(\!|N(s) \vdash x_i|\!), (\!|N(s+1_z) \vdash \nu z P|\!)\rangle \\
&= (\!|N(s) \vdash \alpha.\nu z P|\!)
\end{aligned}$$

(iv) Suppose $\alpha = \tau$

$$\begin{aligned}
(\!|N(s) \vdash \nu z\alpha.P|\!) &= res_s\langle(\!|N(s+1) \vdash \alpha.P|\!)\rangle \\
&= res_s\langle tau_{s+1}\langle(\!|N(s+1) \vdash P|\!)\rangle\rangle \\
\text{by Property 8(a),}\mathbf{R4} &= tau_s\langle res_s\langle(\!|N(s+1) \vdash P|\!)\rangle\rangle \\
&= tau_s\langle(\!|N(s) \vdash \nu z P|\!)\rangle \\
&= (\!|N(s) \vdash \alpha.\nu z P|\!)
\end{aligned}$$

R3 if $z \in subj(\alpha)$ then $SGE \vdash \nu z P = 0$. There are a three cases to check:

(i) Suppose $\alpha = z(z')$

$$\begin{aligned}
(\!|N(s) \vdash \nu z\alpha.P|\!) &= res_s\langle(\!|N(s+1) \vdash \alpha.P|\!)\rangle \\
&= res_s\langle in_{s+1}\langle(\!|N(s+1) \vdash z|\!), \langle(\!|N(s+1) \vdash P[z' := x_j]|\!)\rangle_{j=1}^{n}, \\
&\quad (\!|N(s+1) \vdash P[z' := z]|\!), (\!|N((s+1)+1_z) \vdash P|\!)\rangle \\
\text{by Property 8(a),}\mathbf{R1} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

(ii) Suppose $\alpha = \overline{z}\langle x_i\rangle$

$$\begin{aligned}
(\!|N(s) \vdash \nu z\alpha.P|\!) &= res_s\langle(\!|N(s+1) \vdash \alpha.P|\!) \\
&= res_s\langle out_{s+1}\langle(\!|N(s+1) \vdash z|\!), (\!|N(s+1) \vdash x_j|\!), \\
&\quad (\!|N(s+1) \vdash P|\!)\rangle \\
\text{by Property 8(a),}\mathbf{R2} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

(iii) Suppose $\alpha = \overline{z}(x_i)$

$$\begin{aligned}
(\!|N(s) \vdash \nu z\alpha.P|\!) &= res_s\langle(\!|N(s+1) \vdash \alpha.P|\!) \\
&= res_s\langle bout_{s+1}\langle(\!|N(s+1) \vdash z|\!), (\!|N((s+1)+1_z) \vdash P|\!)\rangle \\
\text{by Property 8(a),}\mathbf{R3} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

R4 if $z \neq x_i$ then $SGE \vdash \nu z\overline{x_i}\langle z\rangle.P = \overline{x_i}(z).P$

$$\begin{aligned}
(\!|N(s) \vdash \nu z\overline{x_i}\langle z\rangle.P|\!) &= res_s\langle(\!|N(s+1) \vdash \overline{x_i}\langle z\rangle.P|\!) \\
&= res_s\langle out_{s+1}\langle(\!|N(s+1) \vdash x_i|\!), (\!|N(s+1) \vdash z|\!), \\
&\quad (\!|N(s+1) \vdash P|\!)\rangle\rangle \\
\text{by Property 8(a),}\mathbf{R2} &= bout_s\langle(\!|N(s) \vdash x_i|\!), (\!|N(s+1) \vdash P|\!)\rangle \\
&= (\!|N(s) \vdash \overline{x_i}(z).P|\!)
\end{aligned}$$

R5 $SGE \vdash \nu z 0 = 0$

$$\begin{aligned}
(\!|N(s) \vdash \nu z 0|\!) &= res_s\langle(\!|N(s+1) \vdash 0|\!)\rangle \\
&= res_s\langle nil_{s+1}\rangle \\
\text{by Property 8(a),}\mathbf{R6} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

4. Parallel

PAR $SGE \vdash P|Q = P\lfloor Q + Q\lfloor P + P||Q$

$$\begin{aligned}
(\!|N(s) \vdash P|Q|\!) &= par_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q|\!)\rangle \\
\text{by Property 8(c),}\mathbf{P1} &= sum_s\langle lm_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q|\!)\rangle, \\
&\quad lm_s\langle (\!|N(s) \vdash Q|\!), (\!|N(s) \vdash P|\!)\rangle, \\
&\quad syn_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q|\!)\rangle\rangle \\
&= sum_s\langle (\!|N(s) \vdash P\lfloor Q|\!), (\!|N(s) \vdash Q\lfloor P|\!), (\!|N(s) \vdash P||Q|\!)\rangle \\
&= (\!|N(s) \vdash P\lfloor Q + Q\lfloor P + P||Q|\!)
\end{aligned}$$

5. Left Merge

LM1 $SGE \vdash 0\lfloor R = 0$

$$\begin{aligned}
(\!|N(s) \vdash 0\lfloor R|\!) &= lm_s\langle (\!|N(s) \vdash 0|\!), (\!|N(s) \vdash R|\!)\rangle \\
&= lm_s\langle nil_s, (\!|N(s) \vdash R|\!)\rangle \\
\text{by Property 8(d),}\mathbf{LM1} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

LM2 $SGE \vdash (P + Q)\lfloor R = P\lfloor R + Q\lfloor R$

$$\begin{aligned}
(\!|N(s) \vdash (P + Q)\lfloor R|\!) &= lm_s\langle (\!|N(s) \vdash P + Q|\!), (\!|N(s) \vdash R|\!)\rangle \\
&= lm_s\langle sum_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q|\!)\rangle, (\!|N(s) \vdash R|\!)\rangle \\
\text{by Property 8(d),}\mathbf{LM2} &= sum_s\langle lm_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash R|\!)\rangle, lm_s\langle (\!|N(s) \vdash Q|\!), \\
&\quad (\!|N(s) \vdash R|\!)\rangle\rangle \\
&= sum_s\langle (\!|N(s) \vdash P\lfloor Q|\!), (\!|N(s) \vdash Q\lfloor R|\!)\rangle \\
&= (\!|N(s) \vdash P\lfloor R + Q\lfloor R|\!)
\end{aligned}$$

LM3 if $bn(\alpha.0) \notin fn(Q)$ then $SGE \vdash (\alpha.P)\lfloor Q = \alpha.(P|Q)$

$$\begin{aligned}
(\!|N(s) \vdash (\alpha.P)\lfloor Q|\!) &= lm_s\langle (\!|N(s) \vdash \alpha.P|\!), (\!|N(s) \vdash Q|\!)\rangle \\
\text{by Property 8(d),}\mathbf{LM3,LM4,LM5,LM6} &= (\!|N(s) \vdash \alpha.(P|Q)|\!)
\end{aligned}$$

6. Synchronization

SYN1 $SGE \vdash 0||P = 0$

$$\begin{aligned}
(\!|N(s) \vdash 0||P|\!) &= syn_s\langle (\!|N(s) \vdash 0|\!), (\!|N(s) \vdash P|\!)\rangle \\
\text{by Property 8(e),}\mathbf{SYN1} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

SYN2 $SGE \vdash P||Q = Q||P$

$$\begin{aligned}
(\!|N(s) \vdash P||Q|\!) &= syn_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q|\!)\rangle \\
\text{by Property 8(e),}\mathbf{SYN2} &= syn_s\langle (\!|N(s) \vdash Q|\!), (\!|N(s) \vdash P|\!)\rangle \\
&= (\!|N(s) \vdash Q||P|\!)
\end{aligned}$$

SYN3 $SGE \vdash (P+Q)||R = P||R + Q||R$

$$\begin{aligned}
(\!|N(s) \vdash (P+Q)||R|\!) &= syn_s\langle (\!|N(s) \vdash P+Q|\!), (\!|N(s) \vdash R|\!)\rangle \\
&= syn_s\langle sum_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash Q|\!)\rangle, (\!|N(s) \vdash P|\!)\rangle \\
\text{by Property 8(e),}\mathbf{SYN3} &= sum_s\langle syn_s\langle (\!|N(s) \vdash P|\!), (\!|N(s) \vdash R|\!)\rangle, syn_s\langle (\!|N(s) \vdash Q|\!), \\
&\quad (\!|N(s) \vdash R|\!)\rangle\rangle \\
&= (\!|N(s) \vdash P||R + Q||R|\!)
\end{aligned}$$

SYN4 $SGE \vdash x_i(z).P||\overline{x_i}\langle x_j\rangle.Q = \tau.(P[z := x_j]||Q)$

$$\begin{aligned}
(\!|N(s) \vdash x_i(z).P||\overline{x_i}\langle x_j\rangle.Q|\!) &= syn_s\langle (\!|N(s) \vdash x_i(z).P|\!), (\!|N(s) \vdash \overline{x_i}\langle x_j\rangle.Q|\!)\rangle \\
&= syn_s\langle in_s\langle (\!|N(s) \vdash x_i|\!), \langle (\!|N(s) \vdash P[z := x_l]|\!)\rangle_{l=1}^{n}, \\
&\quad (\!|N(s+1) \vdash P|\!)\rangle, out_s\langle (\!|N(s) \vdash x_i|\!), (\!|N(s) \vdash x_j|\!), \\
&\quad (\!|N(s) \vdash Q|\!)\rangle\rangle \\
\text{by Property 8(e),}\mathbf{SYN4} &= tau_s\langle par_s\langle (\!|N(s) \vdash P[z := x_j]|\!), (\!|N(s) \vdash Q|\!)\rangle\rangle \\
&= (\!|N(s) \vdash \tau.(P[z := x_j]||Q)|\!)
\end{aligned}$$

SYN5 $SGE \vdash x_i(z).P||\overline{x_i}(z).Q = \tau.(\nu z(P|Q))$

$$\begin{aligned}
(\!|N(s) \vdash x_i(z).P||\overline{x_i}(z').Q|\!) &= syn_s\langle (\!|N(s) \vdash x_i(z).P|\!), (\!|N(s) \vdash \overline{x_i}(z').Q|\!)\rangle \\
&= syn_s\langle in_s\langle (\!|N(s) \vdash x_i|\!), \langle (\!|N(s) \vdash P[z := x_l]|\!)\rangle_{l=1}^{n}, \\
&\quad (\!|N(s+1) \vdash P|\!)\rangle, bout_s\langle (\!|N(s) \vdash x_i|\!), (\!|N(s+1) \vdash Q|\!)\rangle\rangle \\
\text{by Property 8(e),}\mathbf{SYN5} &= tau_s\langle res_s\langle par_{s+1}\langle (\!|N(s+1) \vdash P|\!), (\!|N(s+1) \vdash Q|\!)\rangle\rangle\rangle \\
&= (\!|N(s) \vdash \tau.(\nu z(P|Q))|\!)
\end{aligned}$$

SYN6 $SGE \vdash \tau.P||\alpha.Q = 0$

$$\begin{aligned}
(\!|N(s) \vdash \tau.P||\alpha.Q|\!) &= syn_s\langle (\!|N(s) \vdash \tau.P|\!), (\!|N(s) \vdash \alpha.Q|\!)\rangle \\
&= syn_s\langle tau_s\langle (\!|N(s) \vdash P|\!)\rangle, (\!|N(s) \vdash \alpha.Q|\!)\rangle \\
\text{by Property 8(e),}\mathbf{SYN6} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

SYN7 $SGE \vdash x_i(z).P||x_j(z').Q = 0$

$$\begin{aligned}
(\!|N(s) \vdash x_i(z).P||x_j(z).Q|\!) &= syn_s\langle(\!|N(s) \vdash x_i(z).P|\!), (\!|N(s) \vdash x_j(z).Q|\!)\rangle \\
&= syn_s\langle in_s\langle(\!|N(s) \vdash x_i|\!), \langle(\!|N(s) \vdash P[z := x_l]|\!)\rangle_{l=1}^n, \\
&\quad (\!|N(s+1) \vdash P|\!)\rangle, in_s\langle(\!|N(s) \vdash x_i|\!), \\
&\quad \langle(\!|N(s) \vdash Q[z' = x_l]|\!)\rangle_{l=1}^n, (\!|N(s+1_z) \vdash Q|\!)\rangle\rangle \\
\text{by Property 8(e),}\mathbf{SYN7} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

SYN8 if both α and β are output or bound output prefixes, then $SGE \vdash \alpha.P||\beta.Q = 0$

$$\begin{aligned}
(\!|N(s) \vdash \alpha.P||\beta.Q|\!) &= syn_s\langle(\!|N(s) \vdash \alpha.P|\!), (\!|N(s) \vdash \beta.Q|\!)\rangle \\
\text{by Property 8(e),}\mathbf{SYN8} &= nil_s \\
&= (\!|N(s) \vdash 0|\!)
\end{aligned}$$

7. Equivalence

REF $SGE \vdash P = P$

$$(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash P|\!)$$

SYMM Suppose $SGE \vdash P = Q$ then $SGE \vdash Q = P$

Suppose $(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash Q|\!)$ then it is true by symmetry on the equality that $(\!|N(s) \vdash Q|\!) = (\!|N(s) \vdash P|\!)$.

TRANS Suppose $SGE \vdash P = Q$ and $SGE \vdash Q = R$, then $SGE \vdash P = R$

Suppose that $(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash Q|\!)$ and that $(\!|N(s) \vdash Q|\!) = (\!|N(s) \vdash R|\!)$ then by the transitivity of the equality we have $(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash R|\!)$.

8. Congruences

CON1 Suppose $SGE \vdash P = Q$ then $SGE \vdash \overline{x_i}\langle x_j\rangle.P = \overline{x_i}\langle x_j\rangle.Q$

Suppose that $(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash Q|\!)$ then we have the following equalities:

$$\begin{aligned}
(\!|N(s) \vdash \overline{x_i}\langle x_j\rangle.P|\!) &= out_s\langle(\!|N(s) \vdash x_i|\!), (\!|N(s) \vdash x_j|\!), (\!|N(s) \vdash P|\!)\rangle \\
&= out_s\langle(\!|N(s) \vdash x_i|\!), (\!|N(s) \vdash x_j|\!), (\!|N(s) \vdash Q|\!)\rangle \\
&= (\!|N(s) \vdash \overline{x_i}\langle x_j\rangle.Q|\!)
\end{aligned}$$

CON2 Suppose $SGE \vdash P = Q$ then $SGE \vdash \overline{x_i}(z).P = \overline{x_i}(z).Q$
Suppose that $(\!|N(s+1) \vdash P|\!) = (\!|N(s+1) \vdash Q|\!)$ then we have the following equalities:

$$\begin{aligned}(\!|N(s) \vdash \overline{x_i}(z).P|\!) &= bout_s\langle(\!|N(s) \vdash x_i|\!), (\!|N(s+1) \vdash P|\!)\rangle \\ &= bout_s\langle(\!|N(s) \vdash x_i|\!), (\!|N(s+1) \vdash Q|\!)\rangle \\ &= (\!|N(s) \vdash \overline{x_i}(z).Q|\!)\end{aligned}$$

CON3 Suppose that $\forall i, 1 \leq i \leq n$, $SGE \vdash P[z := x_i] = Q[z := x_j]$ and that $SGE \vdash P = Q$ then $SGE \vdash x_i(z).P = x_i(z).Q$
Suppose that $\forall i, 1 \leq i \leq n$, $(\!|N(s) \vdash P[z := x_i]|\!) = (\!|N(s) \vdash Q[z := x_i]|\!)$ and that $(\!|N(s+1) \vdash P|\!) = (\!|N(s+1) \vdash Q|\!)$ then we have the following equalities:

$$\begin{aligned}(\!|N(s) \vdash x_i(z).P|\!) &= in_s\langle(\!|N(s) \vdash x_i|\!), \langle(\!|N(s) \vdash P[z := x_i]|\!)\rangle_{i=1}^n, (\!|N(s+1) \vdash P|\!)\rangle \\ &= in_s\langle(\!|N(s) \vdash x_i|\!), \langle(\!|N(s) \vdash Q[z := x_i]|\!)\rangle_{i=1}^n, (\!|N(s+1) \vdash Q|\!)\rangle \\ &= (\!|N(s) \vdash x_i(z).Q|\!)\end{aligned}$$

CON4 Suppose $SGE \vdash P = Q$ then $SGE \vdash \tau.P = \tau.Q$
Suppose that $(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash Q|\!)$ then we have the following equalities:

$$\begin{aligned}(\!|N(s) \vdash \tau.P|\!) &= tau_s\langle(\!|N(s) \vdash P|\!)\rangle \\ &= tau_s\langle(\!|N(s) \vdash Q|\!)\rangle \\ &= (\!|N(s) \vdash \tau.Q|\!)\end{aligned}$$

CON5 Suppose $SGE \vdash P = Q$ then $SGE \vdash P + R = Q + R$
Suppose that $(\!|N(s) \vdash P|\!) = (\!|N(s) \vdash Q|\!)$ then we have the following equalities:

$$\begin{aligned}(\!|N(s) \vdash P + R|\!) &= sum_s\langle(\!|N(s) \vdash R|\!), (\!|N(s) \vdash R|\!)\rangle \\ &= sum_s\langle(\!|N(s) \vdash Q|\!), (\!|N(s) \vdash R|\!)\rangle \\ &= (\!|N(s) \vdash Q + R|\!)\end{aligned}$$

CON6 Suppose $SGE \vdash P = Q$ then $SGE \vdash \nu z P = \nu z Q$
Suppose that $(\!|N(s+1) \vdash P|\!) = (\!|N(s+1) \vdash Q|\!)$ then we have the following equalities:

$$\begin{aligned}(\!|N(s) \vdash \nu z P|\!) &= res_s\langle(\!|N(s+1) \vdash P|\!)\rangle \\ &= res_s\langle(\!|N(s+1) \vdash Q|\!)\rangle \\ &= (\!|N(s) \vdash \nu z Q|\!)\end{aligned}$$

CON7 Suppose $SGE \vdash P = Q$ then $SGE \vdash P \downarrow R = Q \downarrow R$
Suppose that $(\![N(s) \vdash P]\!) = (\![N(s) \vdash Q]\!)$ then we have the following equalities:

$$\begin{aligned}(\![N(s) \vdash P \downarrow R]\!) &= lm_s\langle(\![N(s) \vdash P]\!), (\![N(s) \vdash R]\!)\rangle \\ &= lm_s\langle(\![N(s) \vdash Q]\!), (\![N(s) \vdash R]\!)\rangle \\ &= (\![N(s) \vdash Q \downarrow R]\!)\end{aligned}$$

CON8 Suppose $SGE \vdash P = Q$ then $SGE \vdash P || R = Q || R$
Suppose that $(\![N(s) \vdash P]\!) = (\![N(s) \vdash Q]\!)$ then we have the following equalities:

$$\begin{aligned}(\![N(s) \vdash P || R]\!) &= syn_s\langle(\![N(s) \vdash P]\!), (\![N(s) \vdash R]\!)\rangle \\ &= syn_s\langle(\![N(s) \vdash Q]\!), (\![N(s) \vdash R]\!)\rangle \\ &= (\![N(s) \vdash Q || R]\!)\end{aligned}$$

□

Bibliography

[1] S.Abramsky, and A.Jung. *Domain theory* . In **Handbook of Logic in Computer Science**. Oxford University Press, 1994.

[2] Anderson, S.O. and Power, A.J., *A representable approach to finite nondeterminism.* In **Proceedings, MFPS'94, Theoretical Computer Science.** Elsevier.

[3] Barr, M. and Wells, C., *Category Theory for Computing Science.* Prentice Hall International Series in Computer Science, 1990.

[4] Barr, M. and Wells, C., *Toposes, Triples and Theories*, Springer-Verlag, 1985.

[5] Borceaux, F., *Handbook of Categorical Algebra, volume 2.* Cambridge University Press, 1994.

[6] Caccamo, M., Hyland, J. M. and Winskel G., *Lecture Notes in Category Theory [Draft]*, June 2001. To appear in BRICS: Lecture Series.

[7] Cattani, G.L., Stark, I. and Winskel, G., *Presheaf Models for the π-Calculus.* In **Category Theory and Computer Science: Proceedings of the 7th International Conference CTCS '97**, Santa Margherita Ligure, Italy, September 4-6, 1997, **Lecture Notes in Computer Science** 1290, pp. 106–126. Springer-Verlag, 1997.

[8] Day, B. J., *On Closed Categories of Functors.* In **Reports of the Midwest Category Seminar**, Lecture Notes in Mathematics 137, pp. 1–38. Springer-Verlag, 1970.

[9] Fiore, M.P., Moggi, E. and Sangiorgi, D., *A Fully-Abstract Model for the π-calculus.* In **Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science**, pages 43-54, New Brunswick, New Jersey, 27-30 July 1996. IEEE Computer Society Press

[10] Hennesy, M. and Plotkin G., *Full abstraction for a simple parallel programming language.* In J. Becvar, editor, **Proceedings, 8^{th} Symposium on Mathematical Foundations of Computer Science**, volume 74 of **Lecture Notes in Computer Science**. Springer-Verlag, 1979.

[11] Hofmann, M., *Semantical Analysis of Higher-Order Abstract Syntax.* In **Logic in Computer Science (LICS)**. IEEE, Computer Society Press, 1999.

[12] Johnstone, P. T., *Stone Spaces*, Cambridge University Press, 1982.

[13] Lambek, J. and Scott, P., *Introduction to Higher Order Categorical Logic.* Cambridge University Press, 1986.

[14] Mac Lane, S., *Categories for the Working Mathematician.* Springer-Verlag, 1971.

[15] Mendler, N.P., Panangaden, P., Scott, P.J. and Seely R.A.G., *A Logical View of Concurrent Constraint Programming.* In **Nordic Journal of Computing**, Volume 2, pp. 182–221, 1995.

[16] Milner, R., *Communication and Mobile Systems: the π-Calculus.* Cambridge University Press, 1999.

[17] Milner, R., *The polyadic π-calculus: a tutorial.* In **Logic and Algebra of Specification**, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993, pp. 203–246.

[18] Milner, R., Parrow, J. and Walker, D., *A calculus of mobile processes, Parts I and II.* In **Information and Computation**, 100, 1, 1992, pp. 1–77.

[19] O'Hearn, P.W. and Pym, D.J., *The Logic of Bunched Implication.* In **Bulletin of Symbolic Logic** 5, pp. 215–244, 1999.

[20] Pareigis, B., *Categories and functions.* Academic Press, New York, 1970.

[21] J.Parrow, and B.Victor. *The Fusion calculus: expressiveness and symmetry in mobile computing.* In **Symposium on Logic in Computer Science**. IEEE Press, 1998

[22] Sangiorgi, D., *π-calculus, internal mobility and agent-passing calculi.* In **Theoretical Computer Science** 167(2), 1996.

[23] Scott, P. J., *Some Aspects of Categories in Computer Science.* In **Handbook of Algebra**, vol. 2, 2000.

[24] Stark, I., *A Fully Abstract Domain Model for the π-calculus.* In **Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science**, pages 36-42, New Brunswick, New Jersey, 27-30 July 1996. IEEE Computer Society Press.