A Fully-Abstract Model for the π-calculus

M.P. Fiore*[1] and E. Moggi†[2] and D. Sangiorgi‡[3]

*COGS, University of Sussex, †DISI, Universita di Genova, ‡INRIA, Sophia Antipolis

This paper provides both a fully abstract (domain-theoretic) model for the π-calculus and a universal (set-theoretic) model for the finite π-calculus with respect to strong late bisimulation and congruence. This is done by: considering categorical models, defining a metalanguage for these models, and translating the π-calculus into the metalanguage. A technical novelty of our approach is an abstract proof of full abstraction: The result on full abstraction for the finite π-calculus in the set-theoretic model is axiomatically extended to the whole π-calculus with respect to the domain-theoretic interpretation. In this proof, a central role is played by the description of non-determinism as a free construction and by the equational theory of the metalanguage.

INTRODUCTION

The π-calculus [21] is a process algebra for communicating processes with a dynamically-changing topology. *Processes* (or *agents*) interact with each other by exchanging *names*. A name can be *private* (i.e. *local*) to a process, which, however, may decide to export the name thus accepting to share it with other processes. Communication of private names is the main difference between π-calculus and its predecessor CCS, and makes the calculus very expressive (see for instance the encodings of data values [21, 22], and higher-order process calculi [35, 28]). *Late bisimulation* is the operational equivalence on π-calculus processes analysed in [21]. It is preserved by all operators of the calculus except for input. *Late congruence* is the induced congruence. Roughly, in late bisimulation free names of processes are viewed as constants, whereas in late congruence they are viewed as free variables and hence can be freely instantiated.

This paper substantiates the claim that "the π-calculus is CCS with local channels". Indeed, we construct a model for the π-calculus combining techniques used for modelling CCS and local variables in the light of an abstract approach to denotational semantics; notably: *powerdomains* as free algebras, *functor categories*,

[1]Supported by EPSRC grant RR29300 and HCM ERBCHRXCT 92-0046.
[2]Supported by ESPRIT BRA CLICS-II, HCM EXPRESS and SCIENCE ERBSC1*CT920795.
[3]Supported by Working Group CONFER II.

and a kind of *monadic metalanguage*. Powerdomains as free algebras were introduced in [16] for modelling the bounded non-determinism of a parallel imperative language. *Functor categories* were used in [24] to model local variables in Algol-like languages. *Monads* were proposed as a tool for structuring denotational semantics [23, 10].

Using monads we can exhibit the semantics for the π-calculus as a special case of a general construction, which can be instantiated to get semantics for calculi ranging from pure CCS to value-passing CCS to the polyadic π-calculus. This uniform treatment highlights the intrinsic differences and similarities among these calculi. Here we will only present a model for the π-calculus. The model captures essential properties of the role of names in the π-calculus: for instance, that the identity of names does not affect the behaviour of a process and that equality and inequality conditions on names may affect process bisimilarity.

We give a denotational semantics for the π-calculus by: considering categorical models, defining a metalanguage for these models, and translating the π-calculus into the metalanguage. The metalanguage is a simply typed λ-calculus with sums, operations for non-determinism and dynamic allocation, base types for names and agents, and recursion (over exponents of the type of agents). It has an interpretation in a standard *domain-theoretic* model given by a functor category over **Cpo** (the category of cpos —posets closed under lubs of ω-chains possibly without bottom element— and continuous functions) equipped with a powerdomain monad. This model is shown to provide a fully-abstract denotational semantics for the π-calculus with respect to strong late bisimulation (and congruence).

The metalanguage without recursion has an interpretation in a *set-theoretic* model given by a functor category over **Set** (the category of sets and functions) equipped with the free-semilattice monad. This model is in bijective correspondence with certain canonical normal forms of π-calculus processes and provides a *universal* denotational semantics for the finite π-calculus with respect to late bisimulation (and congruence).

An important aspect of the metalanguage is its associated equational theory which permits reasoning about the denotational semantics. For example, to validate laws on π-calculus processes we first validate analogous laws between terms of the metalanguage using its equational theory, and then infer the original laws by compositionality of the denotational interpretation. Also, the denotational semantics implicitly defines a model of *synchronisation trees* for the π-calculus under late bisimulation (i.e. an abstract notion of *late transition system*) and process-like operations on these, whose laws can be established using the equational theory of the metalanguage.

A novelty of our approach is an *abstract proof* of full abstraction. The result on full abstraction for canonical normal forms in the set-theoretic model is axiomatically extended to the whole calculus with respect to the domain-theoretic interpretation. This is done by relating the free-semilattice monad and the powerdomain monad by means of their universal properties, and by exploiting the equational theory of the metalanguage. As a technical benefit, an explicit description of the powerdomain is not needed and we can work with cpos instead of bifinite domains.

In the π-calculus, it is possible to define bisimilarity equivalences in between late bisimilarity and congruence, using *distinctions* [21]. These are conjunctions of inequalities between names which must be respected in any use of the processes. Guided by the denotational model, we have generalised distinctions to *constraints*, roughly decidable properties on finite tuples of names (a similar generalisation is studied, operationally, by Boreale and De Nicola [7]). This extra generality pays off as, for example, for any pair of processes there is an *optimal* constraint which expresses the necessary conditions on names under which the processes are behaviourally equivalent. A full abstraction result for the notion of bisimulation under a constraint (generalising the full abstraction result for late congruence) is also provided.

Related work. In [17] and [13], Hennessy and Plotkin constructed *term models* for CCS-like languages (where actions are pure synchronisations). Later, Abramsky [1] gave a denotational semantics for SCCS using a domain of synchronisation trees defined as the initial solution in **SFP** (the category of bifinite domains and continuous functions) of a domain equation involving Plotkin's powerdomain. Further, he provided two full abstraction results: one for finite SCCS with respect to *strong partial bisimulation* and another one for the whole SCCS with respect to the *finitely observable part of strong bisimulation.* More recently, Aceto and Ingólfsdóttir [4], using the same domain of synchronisation trees, have generalised Abramsky's results to a class of CCS-like languages (those described in the compact GSOS format). Outside the realm of domain theory, in [27], Rutten provides fully abstract semantics with respect to *strong bisimulation* in a non-well-founded set (specified by a recursive equation) for a different class of CCS-like languages (a subclass of those described in the tyft/txft format).

Independently from us, other researchers have been working on a denotational semantics for the π-calculus. Stark has given a denotational semantics for π-calculus in a functor category over **SFP** [34]. His interpretation coincides with ours but, as he has not extracted a metalanguage from the model, his constructions are concrete and do not identify the uniformities that our axiomatic approach highlights. Hennessy [15] combines techniques similar to Stark's with techniques for CCS-like languages with value passing to define denotational models for π-calculus *may* and *must* testing and prove their full abstraction. More recently, Cattani, Stark and Winskel [9] have given a denotational semantics for the π-calculus within an indexed category of profunctors, and have shown that their interpretation is fully abstract in the sense that two processes are bisimilar if and only if so are their denotations with respect to the model-theoretic notion of bisimulation obtained from open maps.

Organisation of the paper. Section 1 recalls the syntax and operational semantics of the π-calculus. Section 2 introduces the system $\mathcal{A}_\pi$ of equational axioms and the ω-birule (all validated by late bisimilarity). These provide methods for establishing equalities between processes and are essential for structuring the proof of full-abstraction. Section 3 establishes the operational validity of the ω-birule for late bisimilarity. Section 4 defines the open and closed interpretation of the π-calculus in a functor category $\mathbf{Cpo}^{\mathcal{I}}$ via translation in a suitable metalanguage. Section 5 establishes the denotational validity of the equational axioms in $\mathcal{A}_\pi$ and the ω-birule. Section 6 proves full abstraction for finite processes, from which one can easily derive full abstraction for the π-calculus, by exploiting the operational and denotational validity of the axioms in $\mathcal{A}_\pi$ and the ω-birule. Section 7 sketches how the denotational semantics can be adjusted to handle bisimilary under constraint. Finally, Section 8 discusses further applications and intrinsic limitations of the techniques used, and suggests directions for future research.

1. π-CALCULUS

We review the syntax and operational semantics of the π-calculus. $\mathcal{N}$ is the countably infinite set of all names, ranged over by a, b, c, d. The class $\mathcal{P}r$ of processes is built from the operators of inaction, sum, matching, mismatching, prefixing, restriction, parallel composition and guarded replication; a prefix can be an input, a free output, a bound output or an silent prefix:

DEFINITION 1.1. (π-calculus, concrete syntax)

$$\begin{aligned} P &:= \mathbf{0} \;\Big|\; P+P \;\Big|\; [a=b]P \;\Big|\; [a\neq b]P \;\Big|\; \alpha.\,P \;\Big| \\ &\qquad \boldsymbol{\nu} a\, P \;\Big|\; P \mid P \;\Big|\; !\alpha.\,P \\ \alpha &:= a(b) \;\Big|\; \overline{a}b \;\Big|\; \overline{a}(b) \;\Big|\; \tau \end{aligned}$$

A process is *finite* when it uses only the operators in the first row.

Remark. It is possible to extend the above definition of finite processes to include the parallel composition and restriction operators (which also preserve finite behaviours). However, this does not add new process behaviours (see the laws of Section 2.3) and complicates some of our proofs.

A restriction $\boldsymbol{\nu} a\, P$ makes name a local to P. An input-prefixed process $a(b).\,P$ waits for a name c to be sent along a and then behaves like $P\{c/b\}$, where $\{c/b\}$ is the substitution of b with c. An output-prefixed process $\overline{a}b.\,P$ sends b along a and then continues like P. A bound output $\overline{a}(b)$ represents the output of a private name b at a; one can think of $\overline{a}(b).\,P$ as an abbreviation for $\boldsymbol{\nu} b \overline{a}b.\,P$. The τ-prefixed process $\tau.\,P$ is capable of evolving to P without interacting with the environment. Sum and parallel composition are used, as in CCS, to express non-determinism and to run two processes in parallel. A matching $[a=b]P$ means "if names a and b are equal then P", a mismatching $[a\neq b]P$ means "if names a and b are not equal then P". A replication $!\alpha.\,P$ represents a countably infinite number of copies of $\alpha.\,P$ in parallel. Guarded replications enjoy simpler algebraic laws than plain replication $!P$. The restriction to guarded replications does not affect expressiveness, since every plain replication can be rewritten in terms of guarded replications, up to strong late congruence [29].

Terminology and notation. We use V to range over finite lists of *distinct names*, which we indicate as $\mathcal{P}_{\text{fin}}(\mathcal{N})$, and $\mathcal{P}r_V$ for the set of processes with free names in V. We write $[a \notin a_0, \dots, a_{n-1}]P$ as an abbreviation for $[a\neq a_0]\dots[a\neq a_{n-1}]P$. For a finite list without repetitions $I = i_0, \dots, i_{n-1}$, we write $\sum_{i\in I} P_i$ as an abbreviation

TABLE 1

The transition system for the π-calculus

$$\texttt{alpha}\quad \frac{P \equiv_\alpha P' \quad P' \xrightarrow{\alpha} P''}{P \xrightarrow{\alpha} P''}$$

`pre`	$\alpha.P \xrightarrow{\alpha} P$	`rep`	$!\alpha.P \xrightarrow{\alpha} P \mid !\alpha.P$ if $\mathrm{bn}(\alpha) \notin \mathrm{fn}(!\alpha.P)$
`sum1`	$\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	`par1`	$\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$ if $\mathrm{bn}(\alpha) \notin \mathrm{fn}(Q)$
`com1`	$\dfrac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}c} Q'}{P \mid Q \xrightarrow{\tau} P'\{c/b\} \mid Q'}$	`close1`	$\dfrac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}(b)} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu} b\,(P' \mid Q')}$
`open`	$\dfrac{P \xrightarrow{\overline{a}b} P'}{\boldsymbol{\nu} b\, P \xrightarrow{\overline{a}(b)} P'}$ if $a \not\equiv b$	`res`	$\dfrac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} a\, P \xrightarrow{\alpha} \boldsymbol{\nu} a\, P'}$ $a \notin \mathrm{names}(\alpha)$
`mch`	$\dfrac{P \xrightarrow{\alpha} P'}{[a=a]P \xrightarrow{\alpha} P'}$	`mismch`	$\dfrac{P \xrightarrow{\alpha} P'}{[a \neq b]P \xrightarrow{\alpha} P'}$ if $a \not\equiv b$

for $P_{i_0} + \ldots + P_{i_{n-1}}$. We use the symbol $\equiv$ for syntactic identity, and $\equiv_\alpha$ for syntactic identity modulo alpha conversion. We assign parallel composition and sum the lowest precedence among the operators. Moreover, $\sum_i P_i + Q$ should be read: $(\sum_i P_i) + Q$. In $a(b).P$, $\boldsymbol{\nu} b\, P$, and $\overline{a}(b).P$ all free occurrences of name b in P are bound. *Free names* (*fn*) and *bound names* (*bn*) of processes and prefixes, name substitution, and alpha conversion are defined as expected. In a statement, we say that a name is *fresh* to mean that it is different from any other name which occurs in the statement or in objects of the statement like processes and substitutions. If α is an input or an output prefix $a(b), \overline{a}b, \overline{a}(b)$, then name a is the *subject* of α, written $\mathrm{subj}(\alpha)$, and $\{a,b\}$ is the set of *names* of α, written $\mathrm{names}(\alpha)$; prefix τ has no prefix and its set of names is empty.

Table 1 shows the transition rules for the calculus. We have omitted the symmetric version of the rules `sum1`, `par1`, `com1`, and `close1`. Process transitions are of the form $P \xrightarrow{\alpha} P'$, and are therefore of four kinds, corresponding to the four kinds of prefixes in the syntax. Input and output transitions describe the interactions that P is willing to undertake with its environment, whereas τ transitions represent an internal activity of P. We refer to [21, 28] for more discussion on the transition rules (however in [21] alpha conversion is not part of the rules) as well as for examples of the expressiveness of the language. The notion of behavioural equivalence proposed for the π-calculus in the original paper [21] is late bisimulation:

DEFINITION 1.2. (Late bisimulation [21]) A relation $\mathcal{R}$ on processes is a *late bisimulation* if $P \mathrel{\mathcal{R}} Q$ implies

1. Whenever $P \xrightarrow{a(b)} P'$ and $b \notin \mathrm{fn}(Q)$, then Q' exists s.t. $Q \xrightarrow{a(b)} Q'$ and for each name c, $P'\{c/b\} \mathrel{\mathcal{R}} Q'\{c/b\}$.

2. Whenever $P \xrightarrow{\alpha} P'$ for $\alpha \equiv \overline{a}b$ or $\alpha \equiv \tau$, then Q' exists s.t. $Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{R} Q'$.

3. Whenever $P \xrightarrow{\overline{a}(b)} P'$ and $b \notin \mathrm{fn}(Q)$, then Q' exists s.t. $Q \xrightarrow{\overline{a}(b)} Q'$ and $P' \mathcal{R} Q'$.

4. The converse of (1-3), on the actions from Q.

Processes P and Q are *late bisimilar*, written $P \sim Q$, if $P \mathcal{R} Q$, for some late bisimulation $\mathcal{R}$.

Late bisimilarity is preserved by all operators except input prefix. The induced congruence, called *late congruence*, is denoted by $\sim^{c}$ and can be defined thus:

DEFINITION 1.3. (Late congruence [21]) Two processes P, Q are *late congruent*, written $P \sim^{c} Q$, if $P\sigma \sim Q\sigma$ for all name substitutions σ.

In the semantics of Section 4, $\sim$ will correspond to the *closed* interpretation — where free names of processes are treated as constants— and $\sim^{c}$ will correspond to the *open* interpretation —where free names are treated as free variables.

2. AN ω-BIRULE AND SOME SYNTACTIC CONSTRUCTIONS

A crucial role in the proof of full abstraction of our denotational semantics is played by a system $\mathcal{A}_\pi$ of axioms and inference rules on π-calculus processes. It allows us to rewrite a process $P \in \mathcal{P}r_V$ into an *expanded form up to any level* n, called $\mathrm{Exp}^n_V(P)$; this is a syntactic form where the operators of parallel composition, restriction and replication may only occur underneath n prefixes. If P is finite then, for any n greater than the number of prefixes in P, process $\mathrm{Exp}^n_V(P)$ contains no parallel compositions, restrictions or replications, and can be thought of a kind of finite synchronisation tree. We call such completely-expanded processes *normal forms.*

The rules of $\mathcal{A}_\pi$ give us a way to effectively compute functions Exp^n_V. With these functions, we can define the following ω-birule, for a given relation "=" on π-calculus processes:

$$\frac{\forall n \quad \mathrm{Nil}^n_V(\mathrm{Exp}^n_V(P_1)) = \mathrm{Nil}^n_V(\mathrm{Exp}^n_V(P_2))}{P_1 = P_2} \; P_1, P_2 \in \mathcal{P}r_V \qquad (1)$$

where $\mathrm{Nil}^n_V(P)$ is obtained from $\mathrm{Exp}^n_V(P)$ by replacing the subprocesses underneath n prefixes with $\mathbf{0}$ (therefore $\mathrm{Nil}^n_V(\mathrm{Exp}^n_V(P))$ is a normal form, for all $P \in \mathcal{P}r_V$). Birule (1) is powerful: comparing a pair of possibly non-finite processes w.r.t. "=" is converted into checking equality between a set of pairs of normal forms (we recall that normal forms are finite processes). If relation "=" has appropriate continuity properties, so to be "approximable", and if the axioms and inference rules in $\mathcal{A}_\pi$ are sound for "=", then also birule (1) is sound for "=".

The full-abstraction proof of our π-calculus model can be thought as divided into three parts:

1. operational validity of the ω-birule (1), where "=" is late bisimilarity;

2. denotational validity of the ω-birule (1), where "=" is the equality induced by the (close) interpretation in the model;

3. full abstraction of the (closed) interpretation on normal forms.

Both parts (1) and (2) require proving the soundness of the rules of $\mathcal{A}_\pi$ for the corresponding semantic equality, and the approximability of this equality. The operational and denotational ω-birules obtained provide methods for establishing equalities between processes which, besides their technical use in proving full abstraction, are of independent interest.

In the remainder of this section we define some syntactic constructions that will be useful for our technical developments: some extra process operators, to make system $\mathcal{A}_\pi$ simpler (Subsection 2.1); some special subsets of processes, including normal and canonical forms (Subsection 2.2); the system of axioms and inference rules $\mathcal{A}_\pi$ and some results for them (Subsection 2.3); some syntactic functions on processes, including the above-mentioned functions Exp_V^n, and the functions CNF_V that map normal forms into canonical normal forms (Subsection 2.4).

2.1. Other process operators

We define two extended class of processes, called $\mathcal{P}r_*$ and $\mathcal{P}r_\perp$. The former has the extra operators *left merge* and *synchronization* (written $\lfloor\!\lfloor$ and $\|$); following the ACP tradition [6], we introduce these operators in order to have a finite set of axioms for parallel composition in system $\mathcal{A}_\pi$. Thus, the grammar for $\mathcal{P}r_*$ is obtained from that for $\mathcal{P}r$ in Definition 1.1 by adding the productions:

$$P := \ldots \;\Big|\; P \lfloor\!\lfloor P \;\Big|\; P \,\|\, P$$

Left merge and synchronisation allow us, intuitively, to decompose the behaviour of a parallel composition $P \mid Q$ into the interactions between P and Q, and the remaining actions. Indeed, synchronisation accounts for the behaviour of $P \mid Q$ given by rules `com1-2` and `close1-2`, whereas left merge accounts for that given by `par1-2`. Their transition rules are

$$\texttt{lm}\ \frac{P \xrightarrow{\alpha} P'}{P \lfloor\!\lfloor Q \xrightarrow{\alpha} P' \mid Q} \ \text{if } \mathrm{bn}(\alpha) \notin \mathrm{fn}(Q)$$

$$\texttt{syn-com1}\ \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}c} Q'}{P \,\|\, Q \xrightarrow{\tau} P'\{c/b\} \mid Q'} \qquad \texttt{syn-close1}\ \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}(b)} Q'}{P \,\|\, Q \xrightarrow{\tau} \nu b\,(P' \mid Q')}$$

plus the symmetric version of `syn-com1` and `syn-close1`. The definitions of late bisimulation and congruence on $\mathcal{P}r_*$ remains as for $\mathcal{P}r$.

Class $\mathcal{P}r_\perp$ has the extra operator *bottom* (written $\perp$); this should be thought of as the undefined processes; it will be used as the syntactic counterpart of the "bottom" of the semantic domain. The grammar for $\mathcal{P}r_\perp$ is obtained from that for $\mathcal{P}r$ in Definition 1.1 by adding the production:

$$P := \ldots \;\Big|\; \perp$$

The constant $\bot$ has no transitions.

2.2. Special subsets of processes

We define, inductively on n, the sets of processes $\mathcal{EX}^n_V$, $\mathcal{E}_\bot\mathcal{X}^n_V$, $\mathcal{NF}^n_V$ and $\mathcal{N}_\bot\mathcal{F}^n_V$; and, from these, the sets $\mathcal{NF}_V$ and $\mathcal{N}_\bot\mathcal{F}_V$. The first four sets (those with an index n) can be defined from the grammar below, for different instantiations of the basic sets $\langle \mathcal{B}_V \mid V \in \mathcal{P}_{\mathrm{fin}}(\mathcal{N})\rangle$. In the grammar for X^n_V, it is assumed that a and b are names in V and that $c \notin V$:

$$\begin{array}{rcl} X^0_V & \in & \mathcal{B}_V \\ X^{n+1}_V & := & X^{n+1}_V + X^{n+1}_V \;\Big|\; \mathbf{0} \;\Big|\; \overline{a}b.\, X^n_V \;\Big|\; \overline{a}(c).\, X^n_{V\cup\{c\}} \;\Big| \\ & & \tau.\, X^n_V \;\Big|\; a(c).\, \Big(\sum_{a\in V}[c=a]X^n_V + [c \notin V]X^n_{V\cup\{c\}}\Big) \end{array} \qquad (2)$$

DEFINITION 2.1. ($\mathcal{EX}^n_V$, $\mathcal{E}_\bot\mathcal{X}^n_V$, $\mathcal{NF}^n_V$, $\mathcal{N}_\bot\mathcal{F}^n_V$, $\mathcal{NF}_V$, $\mathcal{N}_\bot\mathcal{F}_V$) For all $n \in \omega$ and $V \in \mathcal{P}_{\mathrm{fin}}(\mathcal{N})$, we define:

- $\mathcal{EX}^n_V$ is the set of processes generated by symbol X^n_V of grammar (2) when $\mathcal{B}_V = \mathcal{P}r_V$.
- $\mathcal{E}_\bot\mathcal{X}^n_V$ is the set of processes generated by symbol X^n_V of grammar (2) when $\mathcal{B}_V = \mathcal{P}r^V_\bot$.
- $\mathcal{NF}^n_V$ is the set of processes generated by symbol X^n_V of grammar (2) when $\mathcal{B}_V = \{\mathbf{0}\}$.
- $\mathcal{N}_\bot\mathcal{F}^n_V$ is the set of processes generated by symbol X^n_V of grammar (2) when $\mathcal{B}_V = \{\bot\}$.

For all $V \in \mathcal{P}_{\mathrm{fin}}(\mathcal{N})$, we also define:

- $\mathcal{NF}_V \overset{\mathrm{def}}{=} \bigcup_n \mathcal{NF}^n_V$.
- $\mathcal{N}_\bot\mathcal{F}_V \overset{\mathrm{def}}{=} \bigcup_n \mathcal{N}_\bot\mathcal{F}^n_V$.

Set $\mathcal{EX}^n_V$ (resp. $\mathcal{E}_\bot\mathcal{X}^n_V$) collects the processes of $\mathcal{P}r_V$ (resp. $\mathcal{P}r^V_\bot$) which are expanded up to level n, i.e. their first n consecutive actions are explicit. $\mathcal{NF}^n_V$ collects the processes which can perform n consecutive actions at most and which are completed expanded. $\mathcal{NF}_V$ is the set of finite and completed-expanded processes, the V-*normal forms*. $\mathcal{N}_\bot\mathcal{F}^n_V$ and $\mathcal{N}_\bot\mathcal{F}_V$ are similar, but $\bot$ can appear too, at the maximal depth.

We also define the set of *canonical normal forms*; these are processes in normal form that are unique for their equivalence class for late bisimilarity. The canonical forms are very special processes, simpler and easier to manipulate than ordinary ones.

DEFINITION 2.2. ($\mathcal{CNF}_V$) $\mathcal{CNF}_V$, called the *canonical V-normal forms*, is the set of canonical representatives of the equivalence classes obtained by quotienting $\mathcal{NF}_V$ by $\sim$. Formally, $\mathcal{CNF}_V$ is defined from the grammar for $\mathcal{NF}_V$ by imposing a canonical choice for bound names and requiring that, in any term of the form $\sum_i P_i$, processes P_i are ordered lexicographically and are not duplicated (i.e. $P_i \equiv P_j$ implies $i = j$).

The uniqueness of canonical normal forms is expressed by the following proposition:

PROPOSITION 2.1. *For $P, Q \in \mathcal{CNF}_V$, we have: $P \sim Q$ iff $P \equiv Q$.*

We have these containments, for all n and V:

$$\begin{array}{ccccccc} & & \mathcal{N}_\perp\mathcal{F}^n_V & \subset & \mathcal{E}_\perp\mathcal{X}^n_V & \subset & \mathcal{P}r^V_\perp \\ & & & & \cup & & \cup \\ \mathcal{CNF}_V & \subset & \mathcal{NF}_V & \subset & \mathcal{EX}^n_V & \subset & \mathcal{P}r_V \end{array}$$

2.3. The system $\mathcal{A}_\pi$

We introduce the system of axioms and inference rules $\mathcal{A}_\pi$; the axioms are given in Table 2, the inference rules in Table 3. In $\mathcal{A}_\pi$, the axioms for sum, restriction and conditionals are those used in [21] or [25] for axiomatising π-calculus late bisimilarity, and include the semilattice axioms for nil and sum, and some distributivity and cancellation axioms for restriction. There is one axiom for each form of replication in order to push replication inwards, like $!\overline{a}b.\, P = \overline{a}b.\, (P \mid !\overline{a}b.\, P)$. Finally, there are twelve axioms for parallel composition, left merge and synchronisation; they allow us to see the possible derivatives of the parallel composition of two processes whose outermost operator is nil, summation or prefixing. The inference rules are those for equivalence and the congruence rules for prefixing, sum, restriction, left merge and synchronisation; the inference rule for input has multiple premises and uses substitution because late bisimulation is not preserved by input prefix. (We do not need the congruence rules for the other operators; these will actually be inferred from the model.)

We write $\mathcal{A}_\pi \models P = Q$ if $P = Q$ can be inferred from the axioms and rules in $\mathcal{A}_\pi$. We will use also a subset $\mathcal{S}_\pi$ of these rules, which suffices to transform normal forms into canonical normal forms. We first show that the rules in $\mathcal{A}_\pi$ allow us to expand a process up to any level.

LEMMA 2.1. *Let $P \in \mathcal{P}r_V$. Then for all n there is $Q \in \mathcal{EX}^n_V$ s.t. $\mathcal{A}_\pi \models P = Q$.*

Proof. By induction on n. For $n = 0$ there is nothing to prove. For $n > 0$ we proceed by induction on the structure of P. It is convenient to assume that the outermost operator of P may also be left-merge or synchronisation.

Case 1 $P \equiv \mathbf{0}$

$\mathbf{0} \in \mathcal{EX}^n_V$, for all n and V.

Case 2 $P \equiv P_1 + P_2$.

Use the structural induction, and the inference rule `Con5`.

Case 3 $P \equiv \overline{a}b.\, P'$, or $P \equiv \overline{a}(b).\, P'$, or $P \equiv \tau.\, P'$.

Use induction on n and the inference rules `Con1,Con2,Con4`.

Alpha-conv.				
A	If P and Q alpha-convertible then	P	$=$	Q
Summation				
S1		$P + \mathbf{0}$	$=$	P
S2		$P + Q$	$=$	$Q + P$
S3		$P + (Q + R)$	$=$	$(P + Q) + R$
S4		$P + P$	$=$	P
Restriction				
R1		$\boldsymbol{\nu} a\, (P + Q)$	$=$	$\boldsymbol{\nu} a\, P + \boldsymbol{\nu} a\, Q$
R2	if $a \notin \text{names}(\alpha)$ then	$\boldsymbol{\nu} a\, \alpha.\, P$	$=$	$\alpha.\, \boldsymbol{\nu} a\, P$
R3	if $a \equiv \text{subj}(\alpha)$ then	$\boldsymbol{\nu} a\, \alpha.\, P$	$=$	$\mathbf{0}$
R4	if not $a \equiv b$ then	$\boldsymbol{\nu} a\, \overline{b}a.\, P$	$=$	$\overline{b}(a).\, P$
R5		$\boldsymbol{\nu} a\, \mathbf{0}$	$=$	$\mathbf{0}$
Conditionals				
C1		$[a = a]P$	$=$	P
C2	if not $a \equiv b$ then	$[a = b]P$	$=$	$\mathbf{0}$
C3		$[a \neq a]P$	$=$	$\mathbf{0}$
C4	if not $a \equiv b$ then	$[a \neq b]P$	$=$	P
Replication				
Rep	if $\text{bn}(\alpha) \notin \text{fn}(\alpha.\, P)$ then	$!\alpha.\, P$	$=$	$\alpha.\, (P \mid !\alpha.\, P)$
Parallel				
Par		$P \mid Q$	$=$	$P \lfloor\!\lfloor Q + Q \lfloor\!\lfloor P + P \parallel Q$
Left Merge				
LM1		$\mathbf{0} \lfloor\!\lfloor R$	$=$	$\mathbf{0}$
LM2		$(P + Q) \lfloor\!\lfloor R$	$=$	$P \lfloor\!\lfloor R + Q \lfloor\!\lfloor R$
LM3	if $\text{bn}(\alpha) \notin \text{fn}(Q)$ then	$(\alpha.\, P) \lfloor\!\lfloor Q$	$=$	$\alpha.\, (P \mid Q)$
Synchronisation				
Syn1		$\mathbf{0} \parallel P$	$=$	$\mathbf{0}$
Syn2		$P \parallel Q$	$=$	$Q \parallel P$
Syn3		$(P + Q) \parallel R$	$=$	$P \parallel R + Q \parallel R$
Syn4		$a(c).\, P \parallel \overline{b}d.\, Q$	$=$	$[a = b]\tau.\, (P\{{}^{d}\!/\!{}_{c}\} \mid Q)$
Syn5		$a(c).\, P \parallel \overline{b}(c).\, Q$	$=$	$[a = b]\tau.\, \boldsymbol{\nu} c\, (P \mid Q)$
Syn6		$\tau.\, P \parallel \alpha.\, Q$	$=$	$\mathbf{0}$
Syn7		$a(c).\, P \parallel b(c).\, Q$	$=$	$\mathbf{0}$
Syn8	if both α and β are output, then	$\alpha.\, P \parallel \beta.\, Q$	$=$	$\mathbf{0}$

TABLE 2

The axioms of $\mathcal{A}_\pi$

`Equivalence`

`Ref` $P = P$

`Symm` $\dfrac{P = Q}{Q = P}$

`Trans` $\dfrac{P = Q \qquad Q = R}{P = R}$

`Congruence`

`Con1` $\dfrac{P = Q}{\overline{a}b.\, P = \overline{a}b.\, Q}$

`Con2` $\dfrac{P = Q}{\overline{a}(b).\, P = \overline{a}(b).\, Q}$

`Con3` $\dfrac{\forall\, i \in n.\ P\{a_i/b\} = Q\{a_i/b\} \qquad P = Q}{a(b).\, P = a(b).\, Q}\ \mathrm{fn}(a(b).\, P, a(b).\, Q) = \{a_1, \ldots, a_n\}$

`Con4` $\dfrac{P = Q}{\tau.\, P = \tau.\, Q}$

`Con5` $\dfrac{P = Q}{P + R = Q + R}$

`Con6` $\dfrac{P = Q}{\boldsymbol{\nu} a\, P = \boldsymbol{\nu} a\, Q}$

`Con7` $\dfrac{P = Q}{P \lfloor\!\lfloor R = Q \lfloor\!\lfloor R}$

`Con8` $\dfrac{P = Q}{P \parallel R = Q \parallel R}$

TABLE 3

The inference rules of $\mathcal{A}_\pi$

Case 4 $P \equiv a(b).\,P'$.

By alpha conversion, we can assume $b \notin V$. By the induction on n, there are processes $Q_b \in \mathcal{EX}^{n-1}_{V \cup \{b\}}$ and, for all $c \in V$, $Q_c \in \mathcal{EX}^{n-1}_{V}$ s.t.

$$\mathcal{A}_\pi \models P' = Q_b \qquad \mathcal{A}_\pi \models P'\{c\!/\!b\} = Q_c \tag{3}$$

Let $Q \overset{\text{def}}{=} \sum_{c \in V} [b = c]Q_c + [b \notin V]Q_b$. Using the axioms for conditionals `C1-C4` and (3), we can infer

$$\mathcal{A}_\pi \models P'\{c\!/\!b\} = Q\{c\!/\!b\}, \text{ for all } c \in V \tag{4}$$

$$\mathcal{A}_\pi \models P' = Q \tag{5}$$

Hence from `Con3` we can derive

$$\mathcal{A}_\pi \models a(b).\,P' = a(b).\,Q$$

which concludes the case, since $a(b).\,Q \in \mathcal{EX}^n_V$.

Case 5 $P \equiv [a = b]P'$ or $P \equiv [a \neq b]P'$.

Use axioms `C1-C4` to eliminate the outermost conditional and then (possibly) the structural induction on P'.

Case 6 $P \equiv \boldsymbol{\nu} a\, P'$.

By alpha conversion, we can assume $a \notin V$. By structural induction, there is $Q \in \mathcal{EX}^n_{V \cup \{a\}}$ s.t.

$$Q \equiv \sum_{i \in I} \alpha_i.\,Q_i$$

and $\mathcal{A}_\pi \models P' = Q$. Hence, by `Con6`, also $\mathcal{A}_\pi \models \boldsymbol{\nu} a\, P' = \boldsymbol{\nu} a\, Q$. Now, if $I = \emptyset$, then, by `R5`, $\mathcal{A}_\pi \models \boldsymbol{\nu} a\, Q = \mathbf{0}$ and we are done. If $I \neq \emptyset$, then using `R1-R4` and assuming by alpha conversion that $\mathrm{bn}(\alpha_i)$ is fresh, we can derive

$$\mathcal{A}_\pi \models \boldsymbol{\nu} a\, Q = \sum_{i \in I} R_i$$

where, for all i, process R_i is defined as follows:

- $R_i \overset{\text{def}}{=} \mathbf{0}$ if $\mathrm{subj}(\alpha_i) \equiv a$,
- $R_i \overset{\text{def}}{=} \bar{b}(a).\,Q_i$ if $\alpha_i \equiv \bar{b}a$ and $b \neq a$,
- $R_i \overset{\text{def}}{=} \alpha_i.\,\boldsymbol{\nu} a\, Q_i$ otherwise.

If R_i is $\mathbf{0}$, then it is already in $\mathcal{EX}^n_V$. If R_i is a prefixed process, then it can be transformed into a process in $\mathcal{EX}^n_V$ proceeding as in Cases 3 and 4 above.

Case 7 $P \equiv !\alpha.\,P'$.

By alpha conversion, we can assume that the bound name of α, if exists, is fresh. From axiom `Rep` we have

$$\mathcal{A}_\pi \models !\alpha.\,P' = \alpha.\,(P \mid !\alpha.\,P')$$

and then $\alpha.\,(P \mid !\alpha.\,P')$ can then be dealt with in the same way as the prefixed processes of Cases 3 and 4.

Case 8 $P \equiv P_1 \lfloor\!\lfloor P_2$.

By the structural induction on P_1, there is a process of the form $\sum_{i\in I} \alpha_i .\, R_i$ s.t.

$$\mathcal{A}_\pi \models P_1 = \sum_{i\in I} \alpha_i .\, R_i$$

Hence, by rule `Con7`,

$$\mathcal{A}_\pi \models P_1 \lfloor\!\lfloor P_2 = (\sum_{i\in I} \alpha_i .\, R_i) \lfloor\!\lfloor P_2$$

If $I = \emptyset$, then, we can use `LM1` to conclude the proof. Otherwise, by repeatedly applying `LM2`, we have

$$\mathcal{A}_\pi \models (\sum_{i\in I} \alpha_i .\, R_i) \lfloor\!\lfloor P_2 = \sum_{i\in I} ((\alpha_i .\, R_i) \lfloor\!\lfloor P_2)$$

and then, using transitivity, `LM3` and alpha conversion,

$$\mathcal{A}_\pi \models P_1 \lfloor\!\lfloor P_2 = \sum_{i\in I} \alpha_i .\, (R_i \mid P_2)$$

Finally, each $\alpha_i .\, (R_i \mid P_2)$ can be rewritten into a process in $\mathcal{EX}^n_V$ proceeding as in Cases 3 and 4.

Case 9 $P \equiv P_1 \parallel P_2$.

By the structural induction, there are processes $Q_1, Q_2 \in \mathcal{EX}^n_V$ with

$$Q_1 \equiv \textstyle\sum_{i\in I} \alpha_i .\, Q'_i \quad \text{and} \quad Q_2 \equiv \sum_{j\in J} \beta_j .\, Q''_j$$

s.t. $\mathcal{A}_\pi \models P_1 = Q_1$ and $\mathcal{A}_\pi \models P_2 = Q_2$. Hence, using `Syn2` and `Con8`, $\mathcal{A}_\pi \models P_1 \parallel P_2 = Q_1 \parallel Q_2$. Now, if either I or J are empty then, by `Syn1` and `Syn2`, we get $\mathcal{A}_\pi \models Q_1 \parallel Q_2 = \mathbf{0}$ and we are done. Otherwise, by repeatedly applying `Syn2` and `Syn3`, we infer

$$\mathcal{A}_\pi \models P = \sum_{i\in I} \sum_{j\in J} (\alpha_i .\, Q'_i \parallel \beta_j .\, Q''_j)$$

Each term $\alpha_i .\, Q'_i \parallel \beta_j .\, Q''_j$ can be rewritten into a process of the form $\tau .\, R$ or $\mathbf{0}$ using axioms `Syn4`-`Syn8` and `C1`-`C4`. The prefixed processes can be rewritten into process in $\mathcal{EX}^n_V$ proceeding as in Cases 3 and 4.

Case 10 $P \equiv P_1 \mid P_2$.

Using rule `Par`, we have

$$\mathcal{A}_\pi \models P = P_1 \lfloor\!\lfloor P_2 + P_2 \lfloor\!\lfloor P_1 + P_1 \parallel P_2$$

Terms $P_1 \lfloor\!\lfloor P_2$, $P_2 \lfloor\!\lfloor P_1$ and $P_1 \parallel P_2$ can be rewritten into terms in $\mathcal{EX}^n_V$ proceeding as in Cases 7 and 8 above. The results can then be combined into a process in $\mathcal{EX}^n_V$ using the rules for sum.

■

LEMMA 2.2. *For a pair of processes $a(b).P$, $a(b).Q$, and a family of processes $\{P_c, Q_c\}_c$ indexed by a finite set of names V, such that*

- $\mathrm{fn}(a(b).P, a(b).Q) \subseteq V$,
- $b \notin V$, *and*
- $b \notin \mathrm{fn}(P_c, Q_c)$, *for all* $c \in V$,

the inference rule

$$\texttt{DerInp}\ \frac{\forall c \in V.\ P_c = Q_c \qquad P = Q}{\begin{array}{rl} & a(b).\left(\sum_{c\in V}[b=c]P_c + [b \notin V]P\right) \\ = & a(b).\left(\sum_{c\in V}[b=c]Q_c + [b \notin V]Q\right)\end{array}}$$

is derivable in $\mathcal{A}_\pi$.

System $\mathcal{S}_\pi$ collects the axioms for alpha conversion and the monoidal and idempotence laws for sum and rules for equivalence and congruence.

DEFINITION 2.3. $\mathcal{S}_\pi$ is the following set of axioms and rules:

$$\mathcal{S}_\pi \stackrel{\text{def}}{=} \{\texttt{A}, \texttt{S1-S4}, \texttt{Ref}, \texttt{Symm}, \texttt{Trans}, \texttt{Con1}, \texttt{Con2}, \texttt{Con4}, \texttt{Con5}, \texttt{DerInp}\}$$

2.4. Syntactic functions

For all n and V, we define the syntactic functions Exp^n_V, Nil^n_V, Bot^n_V and CNF_V.

Function $\mathrm{Exp}^n_V : \mathcal{P}r_V \to \mathcal{E}\mathcal{X}^n_V$ is s.t. for all $P \in \mathcal{P}r_V$,

$$\mathcal{A}_\pi \models P = \mathrm{Exp}^n_V(P). \tag{6}$$

Lemma 2.1 ensures that such a function exists, and its proof shows us a way of computing it. By definition of Exp^n_V, every interpretation of processes which validates the axioms and rules in $\mathcal{A}_\pi$ will also validate the equation $P = \mathrm{Exp}^n_V(P)$.

Function $\mathrm{Nil}^n_V : \mathcal{E}_\perp\mathcal{X}^n_V \to \mathcal{N}_\perp\mathcal{F}^n_V$ replaces each subcomponent of a process $P \in \mathcal{E}_\perp\mathcal{X}^n_V$ which is underneath n prefixes with the process $\mathbf{0}$. The function is defined inductively on n and, for $n > 0$, inductively on the structure of P:

$$\mathrm{Nil}^0_V(P) \stackrel{\text{def}}{=} \mathbf{0} \tag{7}$$

$$\begin{array}{rcl}
\mathrm{Nil}_V^{n+1}(\bot) & \stackrel{\mathrm{def}}{=} & \bot \\
\mathrm{Nil}_V^{n+1}(\mathbf{0}) & \stackrel{\mathrm{def}}{=} & \mathbf{0} \\
\mathrm{Nil}_V^{n+1}(P+Q) & \stackrel{\mathrm{def}}{=} & \mathrm{Nil}_V^{n+1}(P) + \mathrm{Nil}_V^{n+1}(Q) \\
\mathrm{Nil}_V^{n+1}(\tau.\, P) & \stackrel{\mathrm{def}}{=} & \tau.\, \mathrm{Nil}_V^{n}(P) \\
\mathrm{Nil}_V^{n+1}(\overline{a}b.\, P) & \stackrel{\mathrm{def}}{=} & \overline{a}b.\, \mathrm{Nil}_V^{n}(P) \\
\mathrm{Nil}_V^{n+1}(\overline{a}(b).\, P) & \stackrel{\mathrm{def}}{=} & \overline{a}(b).\, \mathrm{Nil}_{V\cup\{b\}}^{n}(P) \\
\mathrm{Nil}_V^{n+1}\Big(a(b).\, \big(\sum_{c\in V}[b=c]P & \stackrel{\mathrm{def}}{=} & a(b).\, \big(\sum_{c\in V}[b=c]\mathrm{Nil}_V^{n}(P) \\
+[b\notin V]Q \quad \big)\Big) & & +[b\notin V]\mathrm{Nil}_{V\cup\{b\}}^{n}(Q) \quad \big)
\end{array}$$

Function $\mathrm{Bot}_V^n : \mathcal{E}_\bot\mathcal{X}_V^n \to \mathcal{N}_\bot\mathcal{F}_V^n$ is defined like Nil_V^n except for the base case, which is given by $\mathrm{Bot}_V^0(P) = \bot$.

Function $\mathrm{CNF}_V : \mathcal{NF}_V \to \mathcal{CNF}_V$ takes a V-normal form and returns its canonical V-normal form.

LEMMA 2.3. *$F_V^n(G_V^{n+m}(P)) \equiv F_V^n(P)$, when $P \in \mathcal{E}_\bot\mathcal{X}_V^{n+m}$ and $F, G \in \{\mathrm{Nil}, \mathrm{Bot}\}$ and $n, m \geq 0$.*

Proof. By induction on n, using the fact that $\mathcal{E}_\bot\mathcal{X}_V^{n+m} \subseteq \mathcal{E}_\bot\mathcal{X}_V^n$. ■

3. OPERATIONAL VALIDITY OF THE ω-BIRULE

The main result in this section is the operational version of the ω-birule (1), that is

$$\frac{\forall n \quad \mathrm{Nil}_V^n(\mathrm{Exp}_V^n(P_1)) \sim \mathrm{Nil}_V^n(\mathrm{Exp}_V^n(P_2))}{\overline{\qquad P_1 \sim P_2 \qquad}} \; P_1, P_2 \in \mathcal{P}r_V \qquad (8)$$

We shall also prove a syntactic characterisation of $\sim$ on finite process, expressed by the following birule:

$$\frac{\mathrm{CNF}_V(\mathrm{Exp}_V^{n_1}(P_1)) \equiv \mathrm{CNF}_V(\mathrm{Exp}_V^{n_2}(P_2))}{\overline{\qquad P_1 \sim P_2 \qquad}} \; P_1, P_2 \in \mathcal{P}r_V \text{ and finite} \qquad (9)$$

where n_i $(i = 1, 2)$ are any integers greater than the number of prefixes in P_i. This birule will be useful in the proofs of full abstraction of our models on normal forms.

Note that (8) and (9) imply the birule:

$$\frac{\forall n \quad \mathrm{CNF}_V(\mathrm{Nil}_V^n(\mathrm{Exp}_V^n(P_1))) \equiv \mathrm{CNF}_V(\mathrm{Nil}_V^n(\mathrm{Exp}_V^n(P_2)))}{\overline{\qquad P_1 \sim P_2 \qquad}} \; P_1, P_2 \in \mathcal{P}r_V$$

To prove (8) and (9) (in Subsection 3.3), we shall first prove the soundness of the process transformations represented by functions Exp_V^n and CNF_V (Subsection 3.1), and the approximability of late bisimilarity (Subsection 3.2).

3.1. Operational validity of functions $\mathbf{Exp}_V^n$ and $\mathbf{CNF}_V$

System $\mathcal{A}_\pi$ is sound for $\sim$. (It cannot be complete, since $\sim$ is not even semidecidable [32].)

Proposition 3.1. *For $P, Q \in \mathcal{P}r_V$, if $\mathcal{A}_\pi \models P = Q$, then $P \sim Q$.*

Proof. Each axiom and inference rule in $\mathcal{A}_\pi$ can be proved sound for $\sim$ by exhibiting appropriate bisimulation relations. Each case is simple (several of these cases are proved in [21] and [25]). ■

Corollary 3.1 (Operational validity of Exp_V^n). *For $P \in \mathcal{P}r_V$, we have that $P \sim \mathit{Exp}_V^n(P)$*

Proof. From definition of function Exp_V^n and Proposition 3.1. ■

System $\mathcal{S}_\pi$ is sound and complete for $\sim$ over $\mathcal{NF}_V$.

Lemma 3.1. *If $P, Q \in \mathcal{NF}_V$, then $P \sim Q$ iff $\mathcal{S}_\pi \models P = Q$.*

Proof. The implication from right to left holds because of the axioms and rules in $\mathcal{S}_\pi$ are sound for $\sim$.

For the implication from left to right one proceeds by induction on the maximal depth of P and Q, where the depth of a process is its maximal level of nesting of prefixes. If the depth is 0, then $\mathcal{S}_\pi \models P = Q$ can be derived from the rules for sum. Suppose the depth is greater than 0. Then, garbage-collecting $\mathbf{0}$ processes, P and Q are of the form $\sum_{i \in I} \alpha_i . P_i$ and $\sum_{j \in J} \beta_j . Q_j$, respectively. One shows that for all i there is j s.t. $\mathcal{S}_\pi \models \alpha_i . P_i = \beta_j . Q_j$, and the converse, inverting the roles of i and j. Then the thesis follows using the axioms for sum. ■

Lemma 3.2. *If $P \in \mathcal{NF}_V$, then $\mathcal{S}_\pi \models P = \mathit{CNF}_V(P)$.*

Proof. From the definition of canonical normal forms and Lemma 3.1. ■

Corollary 3.2 (Operational validity of function CNF_V). *For $P \in \mathcal{NF}_V$, it holds that $P \sim \mathit{CNF}_V(P)$*

Proof. Follows from Lemma 3.2 and Proposition 3.1, since each rule in $\mathcal{S}_\pi$ is either in $\mathcal{A}_\pi$ or is derivable in $\mathcal{A}_\pi$. ■

3.2. Approximability of late bisimulation

Lemma 3.3. *Let $P \in \mathcal{P}r$ and z a fresh name. If $P \xrightarrow{a(b)} P'$ (resp. $P \xrightarrow{\overline{a}(b)} P'$), then also $P \xrightarrow{a(z)} P''$ (resp. $P \xrightarrow{\overline{a}(z)} P''$) with $P'' \equiv_\alpha P'\{z/b\}$.*

Notation. If $\mathcal{S}$ is a set of processes, then $\{\mathcal{S}\}_{\equiv_\alpha}$ is the quotient of $\mathcal{S}$ w.r.t. $\equiv_\alpha$ (alpha conversion).

LEMMA 3.4. *Let $P \in \mathcal{P}r$. Then for all α, the set*

$$\{P' \text{ s.t. } P \xrightarrow{\alpha} P'\}_{\equiv_\alpha}$$

is finite.

Proof. By structural induction. If α is a visible action, then the thesis is easy. The interesting case for $\alpha \equiv \tau$ is parallel composition. Thus, suppose $P \equiv P_1 \mid P_2$. By the inductive hypothesis, up to alpha conversion, the set of the processes R s.t. $P_1 \mid P_2 \xrightarrow{\tau} R$ is derived with an inference proof in which last rule applied is `par1` (or its symmetric version) is finite. We have to prove that the same holds for those transitions $P_1 \mid P_2 \xrightarrow{\tau} R$ whose inference proof uses `close1`, `com1` (or their symmetric versions) in the last step. We only look at `close1`, since the other cases are similar.

For a name u, let S_u be the set of processes R st $P_1 \mid P_2 \xrightarrow{\tau} R$ can be inferred with a proof whose last step is of the form

$$\frac{P_1 \xrightarrow{a(u)} P_1' \quad P_2 \xrightarrow{\overline{a}(u)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} R \stackrel{\text{def}}{=} \boldsymbol{\nu} u\,(P_1' \mid P_2')}$$

for some P_1', P_2' and a.

Now, if z is a fixed fresh name, then, by the assertion of this lemma on visible actions, the sets

$$\{P_1' \text{ s.t. } P_1 \xrightarrow{a(z)} P_1'\}_{\equiv_\alpha}$$
$$\{P_2' \text{ s.t. } P_2 \xrightarrow{a(z)} P_2'\}_{\equiv_\alpha}$$

are finite. Therefore the set S_z must be finite too.

We now show that for any other name w and set S_w, each process $R' \in S_w$ is alpha convertible to a process in S_z. This would conclude the case of rule `close1` and the proof of this lemma. The last step in the inference proof for $P_1 \mid P_2 \xrightarrow{\tau} R'$ is of the form:

$$\frac{P_1 \xrightarrow{a(w)} P_1'' \quad P_2 \xrightarrow{\overline{a}(w)} P_2''}{P_1 \mid P_2 \xrightarrow{\tau} R' \stackrel{\text{def}}{=} \boldsymbol{\nu} w\,(P_1'' \mid P_2'')}$$

By Lemma 3.3, we also have

$$\frac{P_1 \xrightarrow{a(z)} P_1' \quad P_2 \xrightarrow{\overline{a}(z)} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} \boldsymbol{\nu} z\,(P_1' \mid P_2')}$$

with $P_i' \equiv_\alpha P_i''\{z/w\}$, $i = 1, 2$. Moreover, $\boldsymbol{\nu} z\,(P_1' \mid P_2') \in S_z$ and

$$\boldsymbol{\nu} z\,(P_1' \mid P_2') \equiv_\alpha \boldsymbol{\nu} w\,(P_1'\{w/z\} \mid P_2'\{w/z\}) \equiv_\alpha R'.$$

■

DEFINITION 3.1. (Approximations for $\sim$) We define $\sim_{\rm n} \subseteq \mathcal{P}r \times \mathcal{P}r$ by induction on n:

- $\sim_0 \stackrel{\rm def}{=} \mathcal{P}r \times \mathcal{P}r$.
- For $n > 0$, $P \sim_{\rm n} Q$ if:

   1. Whenever $P \xrightarrow{a(b)} P'$ and $b \notin \mathrm{fn}(Q)$, then Q' exists s.t. $Q \xrightarrow{a(b)} Q'$ and for each name c, $P'\{c/b\} \sim_{\rm n-1} Q'\{c/b\}$.
   2. Whenever $P \xrightarrow{\alpha} P'$ for $\alpha \equiv \overline{a}b$ or $\alpha \equiv \tau$, then Q' exists s.t. $Q \xrightarrow{\alpha} Q'$ and $P' \sim_{\rm n-1} Q'$.
   3. Whenever $P \xrightarrow{\overline{a}(b)} P'$ and $b \notin \mathrm{fn}(Q)$, then Q' exists s.t. $Q \xrightarrow{\overline{a}(b)} Q'$ and $P' \sim_{\rm n-1} Q'$.
   4. The converse of (1-3), on the actions from Q.

DEFINITION 3.2. We set $P \sim_\omega Q$ if, for all n, $P \sim_{\rm n} Q$.

LEMMA 3.5. *For all n, it holds that $\sim \subseteq \sim_{\rm n}$.*

LEMMA 3.6. *If $P \sim_{\rm n} Q$ for infinitely many n, then $P \sim_\omega Q$.*

PROPOSITION 3.2. *Relations $\sim_\omega$ and $\sim$ coincide.*

Proof. The inclusion $\sim \subseteq \sim_\omega$ is given by Lemma 3.5. For the opposite inclusion, we use Lemma 3.6 and show that

$$\mathcal{R} \stackrel{\rm def}{=} \{(P, Q) \text{ s.t. } P \sim_{\rm n} Q \text{ for infinitely many } n\}$$

is a late bisimulation. Suppose that $P \mathcal{R} Q$ and $P \xrightarrow{\alpha} P'$ with $\mathrm{bn}(\alpha) \notin \mathrm{fn}(Q)$. We show that Q can match this action. We only consider the case when α is an input, for the other cases are simpler.

By Lemma 3.4, up to alpha conversion there are only a finite number of processes $Q_1, \ldots, Q_m$ s.t. $Q \xrightarrow{\alpha} Q_i$. Moreover, by definition of $\sim_{\rm n}$, if $P \sim_{\rm n} Q$ and $n > 0$, then there is Q_i, $1 \leq i \leq m$, s.t. if $b \equiv \mathrm{bn}(\alpha)$, then

$$\text{for all } c,\ P'\{c/b\} \sim_{\rm n-1} Q_i\{c/b\}. \tag{10}$$

Since $P \sim_{\rm n} Q$ for infinitely many n and $\{Q_1, \ldots, Q_m\}$ is finite, there must be at least a Q_i s.t. (10) holds for infinitely many n. Using this Q_i, process Q can match the action $P \xrightarrow{\alpha} P'$. ■

3.3. The core of the proofs

Lemma 3.7. *For $P, Q \in \mathcal{NF}^n_V$, it holds that $P \sim_{\mathrm{n}} Q$ iff $P \sim Q$.*

Lemma 3.8. *For $P \in \mathcal{EX}^n_V$, it holds that $\mathrm{Nil}^n_V(P) \sim_{\mathrm{n}} P$.*

Proposition 3.3. *For all $P, Q \in \mathcal{P}r_V$, it holds that:*

$$P \sim_{\mathrm{n}} Q \textit{ iff } \mathrm{Nil}^n_V(\mathit{Exp}^n_V(P)) \sim \mathrm{Nil}^n_V(\mathit{Exp}^n_V(Q)).$$

Proof. By Corollary 3.1 and the inclusion $\sim \subseteq \sim_{\mathrm{n}}$, we have $P \sim_{\mathrm{n}} \mathrm{Exp}^n_V(P)$ and $Q \sim_{\mathrm{n}} \mathrm{Exp}^n_V(Q)$. Hence,

$$P \sim_{\mathrm{n}} Q \;\text{ iff }\; \mathrm{Exp}^n_V(P) \sim_{\mathrm{n}} \mathrm{Exp}^n_V(Q)$$

Using Lemmas 3.7 and 3.8 and similar reasoning,

$$\mathrm{Exp}^n_V(P) \sim_{\mathrm{n}} \mathrm{Exp}^n_V(Q) \;\text{ iff }\; \mathrm{Nil}^n_V(\mathrm{Exp}^n_V(P)) \sim \mathrm{Nil}^n_V(\mathrm{Exp}^n_V(Q))$$

■

Proposition 3.4. *For all $P, Q \in \mathcal{NF}_V$, it holds that $P \sim Q$ iff $\mathit{CNF}_V(P) \equiv \mathit{CNF}_V(Q)$*

Proof. Follows from Corollary 3.2 and Proposition 2.1. ■

Theorem 3.1 (Operational validity of the ω-birule). *Birule (8) is valid, for all V.*

Proof. If $P, Q \in \mathcal{P}r_V$, we have:

$$\begin{array}{lll} P \sim Q & \text{iff } \forall n \;\; P \sim_{\mathrm{n}} Q & \text{, by Proposition 3.2} \\ & \text{iff } \forall n \;\; \mathrm{Nil}^n_V(\mathrm{Exp}^n_V(P)) \sim \mathrm{Nil}^n_V(\mathrm{Exp}^n_V(Q)) & \text{, by Proposition 3.3} \end{array}$$

■

Lemma 3.9. *Suppose that $P \in \mathcal{P}r_V$ is finite, and that n is greater than the number of prefixes in the definition of P. Then $\mathit{Exp}^n_V(P) \in \mathcal{NF}_V$.*

Proof. By definition of Exp^n_V, it holds that $\mathcal{A}_\pi \models P = \mathrm{Exp}^n_V(P)$ and that $\mathrm{Exp}^n_V(P) \in \mathcal{EX}^n_V$. Process P can perform $n-1$ consecutive actions at most. Since $P \sim \mathrm{Exp}^n_V(P)$ (Corollary 3.1), also $\mathrm{Exp}^n_V(P)$ can perform $n-1$ consecutive actions at most. Therefore, the depth of $\mathrm{Exp}^n_V(P)$ is $n-1$ at most. By definitions of $\mathcal{EX}^n_V$ and $\mathcal{NF}^n_V$ (Definition 2.1), any process in $\mathcal{EX}^n_V$ whose depth is less than n is in $\mathcal{NF}^n_V$; hence it is also in $\mathcal{NF}_V$. ■

LEMMA 3.10. *Suppose that $P \in \mathcal{P}r_V$ is finite, and that n is greater than the number of prefixes in the definition of P. Then $\mathcal{A}_\pi \models P = CNF_V(Exp^n_V(P))$.*

Proof. The application $\mathrm{CNF}_V(\mathrm{Exp}^n_V(P))$ makes sense because, by Lemma 3.9, $\mathrm{Exp}^n_V(P) \in \mathcal{NF}_V$, and $\mathcal{NF}_V$ is the domain of function CNF.

By definition of Exp^n_V, it holds that $\mathcal{A}_\pi \models P = \mathrm{Exp}^n_V(P)$. Since $\mathrm{Exp}^n_V(P) \in \mathcal{NF}_V$, by Lemma 3.2

$$\mathcal{A}_\pi \models \mathrm{Exp}^n_V(P) = \mathrm{CNF}_V(\mathrm{Exp}^n_V(P)).$$

The thesis of the lemma then follows by transitivity. ■

Lemma 3.10 allows us to strengthen Proposition 3.4, about normal forms, to finite processes:

THEOREM 3.2 (Syntactic characterisation of $\sim$ on finite process). *Birule (9) is valid, for all V.*

Proof. As in the assertion of the previous lemma, the applications $\mathrm{CNF}_V(\mathrm{Exp}^{n_i}_V(P))$ makes sense because, by Lemma 3.9, $\mathrm{Exp}^{n_i}_V(P) \in \mathcal{NF}_V$.

By Lemma 3.10 and Corollary 3.1, $P_1 \sim P_2$ iff $\mathrm{CNF}_V(\mathrm{Exp}^{n_1}_V(P_1)) \sim \mathrm{CNF}_V(\mathrm{Exp}^{n_2}_V(P_2))$. Since canonical normal forms are unique (Proposition 2.1), the latter is true iff $\mathrm{CNF}_V(\mathrm{Exp}^{n_1}_V(P_1)) \equiv \mathrm{CNF}_V(\mathrm{Nil}^{n_2}_V(P_2))$. ■

4. DENOTATIONAL SEMANTICS

We introduce a semantic universe and equip it with constructors for modelling non-determinism and dynamic allocation, and objects of names and agents. These model-theoretic considerations suggest a metalanguage which is presented next. By translation into the metalanguage, we give open and closed interpretations of the π-calculus respectively corresponding to late congruence and bisimulation.

4.1. The semantic universe

The denotation of a process is given relative to the names which are free (or visible) in it. Thus, our model is a type A of agents which *varies over stages*; intuitively, the number of free names available for interaction. That is, the type A consists of the following data: for every natural number n a type $A(n)$ of 'processes with at most n free names' and for every injective substitution $\iota : n \hookrightarrow m$ of n names into m names a mapping $A(n) \xrightarrow{A(\iota)} A(m)$ which allows us to view every process with n free names as a process with m free names. Of course, these mappings cannot be arbitrary: we expect that $A(\mathrm{id}_n) = \mathrm{id}_{A(n)}$ as the substitution $\mathrm{id}_n : n \hookrightarrow n$ produces no renaming; while for injective substitutions $\iota : i \hookrightarrow j$ and $\kappa : j \hookrightarrow k$ we expect that the mapping $A(i) \xrightarrow{A(\kappa\iota)} A(j)$ induced by $\kappa\iota$ decomposes as the mappings $A(i) \xrightarrow{A(\iota)} A(j) \xrightarrow{A(\kappa)} A(k)$ induced by ι and κ. All this amounts to saying that A is a functor from a category $\mathcal{I}$ (of injective maps between finite cardinals) to a category $\mathcal{C}$ (of meanings).

Henceforth we will take the viewpoint that A is a type in the functor category $\mathcal{C}^{\mathcal{I}}$ whose objects are functors from $\mathcal{I}$ to $\mathcal{C}$ and whose arrows are natural transformations between such functors. (Recall that a natural transformation $\varphi : X \to Y$ between functors $X, Y : \mathcal{I} \to \mathcal{C}$ is given by a family $\{\varphi_n : X(n) \to Y(n)\}_{n \in |\mathcal{I}|}$ of morphisms in $\mathcal{C}$ such that for all injective maps $\iota : n \hookrightarrow m$, $Y(\iota) \circ \varphi_n = \varphi_m \circ X(\iota)$.) A process with no free names will have a denotation as a global element of A (i.e. a natural transformation $1 \dot{\to} A$) or, equivalently, as an element of $A(0)$ because, by Yoneda, every $p : A$ is uniquely determined by $p_0 : A(0)$ as $p_n = A(0 \hookrightarrow n)(p_0) : A(n)$. Thus, the naturality condition imposes a *uniform* interpretation at all stages. To interpret arbitrary processes we will introduce a type N of names and assign to a process with n free names a denotation as a natural transformation $N^n \dot{\to} A$. As before the naturality condition imposes a uniform interpretation which, for example, will account for the irrelevance of the identity of names.

It is possible to *axiomatise* the structure needed from our category $\mathcal{C}$ of meanings in order to support the denotational semantics, in particular $\mathcal{C}$ must be a **Cpo**-enriched bicartesian closed category (**Cpo**-biCCC). We will not give an explicit axiomatisation of $\mathcal{C}$ rather we will only be concerned with two categories $\mathcal{C}$ of meanings; viz. **Cpo** and **Set**. In $\mathbf{Cpo}^{\mathcal{I}}$ we will obtain a domain-theoretic model for the whole π-calculus; while in $\mathbf{Set}^{\mathcal{I}}$ we will obtain a set-theoretic model for the finite processes.

4.2. Non-determinism

We introduce the *power* type constructor P for modelling non-determinism. It has associated operations $val_X : X \Rightarrow PX$ and $let_{X,Y} : (X \Rightarrow PY) \times PX \Rightarrow PY$ (roughly, $let(f, p)$ is '$\bigcup\{f(x) \mid x \in p\}$') subject to the laws for a *commutative* monad (see (12) below) and operations $0_X : PX$ and $\cup_X : PX \times PX \Rightarrow PX$ making $(PX, 0_X, \cup_X)$ into a semilattice (see (13) below). Moreover, the operations for non-determinism "commute" with *let* (see (14) below).

In general these power types arise as free constructions by considering left adjoints to forgetful functors from a category of non-deterministic computations to a category of values (see [16, 5]). Two such constructions that will be used later are:

- *The free-semilattice monad.* Let $\mathcal{S}$ be a **Cpo**-cartesian category, and let $\mathbf{SL}(\mathcal{S})$ be the **Cpo**-category of semilattices in $\mathcal{S}$ and homomorphisms. The free-semilattice monad on $\mathcal{S}$ is the **Cpo**-monad induced by the **Cpo**-adjunction $\begin{tikzcd}\mathbf{SL}(\mathcal{S}) \arrow[r, shift right] \arrow[r, phantom, "\perp"] & \mathcal{S} \arrow[l, shift right]\end{tikzcd}$ whenever it exists. When $\mathcal{S} = \mathbf{Set}$ the free-semilattice monad is the *finite* powerset functor equipped with the singleton and the *big* union.

- *Abramsky's powerdomain monad.* Let $\mathcal{D}$ be a **Cpo**-cartesian category. We define the category $\mathbf{ND}(\mathcal{D})$ of non-deterministic computations over $\mathcal{D}$ as the **Cpo**-category with objects $(D, \bot, 0, \cup)$, where $\bot : D$ is the *least element* of D and $(D, 0, \cup)$ is a semilattice in $\mathcal{D}$; arrows are *strict* semilattice homomorphisms.

  Abramsky's Powerdomain monad on $\mathcal{D}$ is the **Cpo**-monad on $\mathcal{D}$ induced by the **Cpo**-adjunction $\begin{tikzcd}\mathbf{ND}(\mathcal{D}) \arrow[r, shift right] \arrow[r, phantom, "\perp"] & \mathcal{D} \arrow[l, shift right]\end{tikzcd}$ whenever it exists.

  When $\mathcal{D} = \mathbf{Cpo}$ Abramsky's powerdomain monad exists (see [16, 3]).

Below we will be concerned with commutative monads for non-determinism arising as above over functor categories. The following observation incorporates them in our setting:

PROPOSITION 4.1. *If $\mathcal{C}$ admits Abramsky's powerdomain monad (resp. the free-semilattice monad) then so does the functor category $\mathcal{C}^{\mathcal{W}}$ for every small category $\mathcal{W}$ and it is given pointwise. That is, writing P and $P^{\mathcal{W}}$ for the monad on $\mathcal{C}$ and $\mathcal{C}^{\mathcal{W}}$ respectively, we have $P^{\mathcal{W}}(X)(w) = P(X(w))$.*

4.3. Dynamic allocation

We introduce the type constructor δ for modelling dynamic allocation of names. It has associated operations $\delta_{X,Y} : (X \Rightarrow Y) \Rightarrow \delta X \Rightarrow \delta Y$, $\mathsf{up}_X : X \Rightarrow \delta X$, and $\mathsf{swap}_X : \delta^2 X \Rightarrow \delta^2 X$; which are the internal version of categorical structure on $\mathcal{C}^{\mathcal{I}}$ (made explicit below) induced by $\mathcal{I}$.

Intuitively δX stands for the type X when one *new* name is available. With this in mind,

- $\mathsf{up}_{X,n}$ is the canonical *coercion* mapping an element of X at stage n to the *same* element at stage $n+1$;
- $\mathsf{swap}_{X,n}$ is an involution mapping an element of X at stage $n+2$ to the element of the same type where the last two names in $n+2$ have been swapped.

To explain how such structure is obtained, we consider an alternative characterisation of the category $\mathcal{I}$ of finite cardinals $n = \{0, \ldots, n-1\}$ and injective maps:

$\mathcal{I}$ is the strict monoidal category $(\mathcal{I}, 0, +)$ freely generated from one object 1 and morphisms $up : 0 \to 1$ and $swap : 2 \to 2$ (here $2 = 1 + 1$) such that

$$\left.\begin{array}{l} swap \circ swap = \mathrm{id}_2 : 2 \to 2 \\ (up + \mathrm{id}_1) \circ up = (\mathrm{id}_1 + up) \circ up : 0 \to 2 \\ swap \circ (up + \mathrm{id}_1) = (\mathrm{id}_1 + up) : 1 \to 2 \end{array}\right\} \tag{11}$$

The functor $\delta : \mathcal{C}^{\mathcal{I}} \to \mathcal{C}^{\mathcal{I}}$, and the natural transformations $\mathsf{up} : \delta^0 \dot{\to} \delta$ and $\mathsf{swap} : \delta^2 \dot{\to} \delta^2$ are defined in terms of the generating data $(1, up, swap)$ for $\mathcal{I}$

- $(\delta X)(n) \overset{\mathrm{def}}{=} X(n+1)$
- $\mathsf{up}_{X,n} \overset{\mathrm{def}}{=} X(n + up) : X(n) \to X(n+1)$
- $\mathsf{swap}_{X,n} \overset{\mathrm{def}}{=} X(n + swap) : X(n+2) \to X(n+2)$

and inherit the equational properties of *up* and *swap* (see (15) below). Moreover, δ, up, and swap preserve anything *defined pointwise* (e.g. see (18) below); such as limits and colimits, and Abramsky's powerdomain monad with its associated structure ($\bot$, 0, and $\cup$).

The functor δ has a tensorial strength $\mathsf{up}_X \times \mathrm{id}_{\delta Y} : X \times \delta Y \to \delta(X \times Y)$ —here we use that $\delta(X \times Y) = \delta(X) \times \delta(Y)$— which, as usual in the presence of exponentials, allow us to internalise the action of δ on morphisms as a map $\delta_{X,Y} : (X \Rightarrow Y) \Rightarrow \delta X \Rightarrow \delta Y$ obeying functorial laws (see (16) below).

4.4. Names and agents

In $\mathcal{C}^{\mathcal{I}}$ we define the objects N of names and A of agents with their associated operations.

Names

The type of names N is defined as the *inclusion* of $\mathcal{I}$ into the biCCC $\mathcal{C}$; i.e. $N(n) \overset{\text{def}}{=} n$ (on the r.h.s. n is the coproduct in $\mathcal{C}$ of n copies of the terminal object). The type N admits a decidable equality $\mathsf{eq} : N \times N \Rightarrow 2$ (here $2 = 1 + 1$) and satisfies the following properties which are used for interpreting the π-calculus:

- There is a global element $\mathsf{new} : \delta N$ (viz. $\mathsf{new}_n \overset{\text{def}}{=} n$) making $N \xrightarrow{\mathsf{up}_N} \delta N \xleftarrow{\mathsf{new}} 1$ into a coproduct diagram in $\mathcal{C}^{\mathcal{I}}$ (see (21) below). Thus we can do case analysis on δN.
- swap_N *swaps* δ new and δ up_N new (see (22) below).

We will also use the following proposition stating that an element of $N \Rightarrow X$ at stage n is determined by n elements of X at stage n together with an element of δX at stage n.

Proposition 4.2. *For any biCCC $\mathcal{K}$, the exponential* $\mathsf{Eval} : (N{\Rightarrow}X) \times N \to X$ *in $\mathcal{K}^{\mathcal{I}}$ exists and is given by*

- $(N \Rightarrow X)(n) = X(n)^n \times X(n+1)$;
- *for an inclusion $\iota = (n \subseteq n+1)$, $(N \Rightarrow X)(\iota)$ is the composite*

$$\begin{tikzcd}
X(n)^n{\times}X(n+1) \arrow[rr] \arrow[dr, "X(\iota)^n{\times}\langle \mathrm{id}, X(\iota+1)\rangle"'] & & X(n+1)^{n+1} \times X(n+2) \\
& X(n+1)^n{\times}X(n+1){\times}X(n+2) \arrow[ur, "\cong"'] &
\end{tikzcd}$$

for an iso $\iota : n \cong n$, $(N \Rightarrow X)(\iota)$ is

$$X(n)^n{\times}X(n+1) \xrightarrow{X(\iota)^\iota{\times}X(\iota+1)} X(n)^n{\times}X(n+1) \quad ,$$

and the above determine the action of $N \Rightarrow X$ on morphisms because every morphism in $\mathcal{I}$ is a composite of inclusions followed by an iso;
- *for $n \in |\,\mathcal{I}\,|$, Eval_n is the composite*

$$X(n)^n \times X(n+1) \times n \xrightarrow{\pi_1 \times \mathrm{id}} X(n)^n \times n \xrightarrow{\mathsf{eval}} X(n)$$

Moreover, the exponential transpose of $\tau : Y \times N \to X$ is $\lambda\tau : Y \to N \Rightarrow X$ given by

$$(\lambda\tau)_n \quad = \quad (\ Y(n) \xrightarrow{\langle \lambda(\tau_n), \tau_{n+1} \circ \langle Y(\iota_n), \kappa_n\rangle\rangle} X(n)^n \times X(n+1)\ )$$

where ι_n is the inclusion $(n \subseteq n+1)$ and κ_n is the constantly n map $Y(n) \to n+1$.

Remark. Note that the natural transformation $N \Rightarrow X \dot{\to} \delta X$ interpreting the judgement $f : N \Rightarrow X \vdash \delta\ f\ \mathsf{new} : \delta X$ is, at stage n, the $(n+1)$-projection map $\pi_{n+1} : X(n)^n \times X(n+1) \to X(n+1)$.

Agents

The type of agents A is defined as $\mu X.\, P(HX)$ where

- $HX \overset{\text{def}}{=} N \times (N \Rightarrow X) + N \times N \times X + N \times \delta X + X$; each summand respectively giving meaning to inputs, free outputs, bound outputs, and internal actions of processes; and
- in the set-theoretic interpretation P is the free-semilattice monad on $\mathbf{Set}^{\mathcal{I}}$, whilst in the domain-theoretic interpretation it is Abramsky's powerdomain monad on $\mathbf{Cpo}^{\mathcal{I}}$. Here one can apply the standard techniques for solving recursive domain equations —see [33] and [11, Chapter 7].

By Propositions 4.1 and 4.2, and results on the solution of domain equations, we can give a concrete description of A on objects as the initial solution to the following system of equations in $\mathcal{C}$ (where P below is the appropriate power monad on $\mathcal{C}$) with $n \in |\,\mathcal{I}\,|$

$$X_n = P(n \times {X_n}^n \times X_{n+1} + n \times n \times X_n + n \times X_{n+1} + X_n).$$

4.5. A metalanguage

We introduce a metalanguage (i.e. a type theory with an associated equational theory) which is an internal language for the part of $\mathcal{C}^{\mathcal{I}}$ we are interested in. This will allow us to perform constructions and definitions in $\mathcal{C}^{\mathcal{I}}$ and prove properties (e.g. about the equality of certain morphisms) without the need to look at explicit descriptions.

The core of the metalanguage is a simply typed λ-calculus (1, $\times$, $\Rightarrow$) with sums ($+$), type constructors for non-determinism (P) and dynamic allocation of names (δ), and base types for names (N) and agents (A). In addition, in the domain-theoretic interpretation, we have operators for recursion over exponents of the type of agents (see (23) below).

Notation. In expressions of the metalanguage we adopt the following conventions: e is an expression, x, y, z are variables, f, g, h are variables of functional type, p, q are variables of computational type.

- The *simple types* are 1 and, for X and Y types, $X \times Y$, $X \Rightarrow Y$, and $X + Y$. We use a standard notation for their constructors and destructors except for $\mathsf{case}\ e\ \mathsf{of}\ \mathsf{in}_1(x)\ \texttt{=>}\ e_1\ \mathsf{or}\ \mathsf{in}_2(y)\ \texttt{=>}\ e_2$ which, to improve readability, we write as $\begin{bmatrix} \lambda x : A.\, e_1 \\ \lambda y : B.\, e_2 \end{bmatrix} e$.

For convenience, for types X, Y, and Z, we will consistently assume the following type equalities:

$$1 \Rightarrow X = X; \qquad X \Rightarrow 1 = 1;$$
$$(X \star Y) \star Z = X \star (Y \star Z), \text{ for } \star \in \{\times, +\}$$

where for either type in the last equality we write $X \star Y \star Z$.

- For X a type, we have a *power type* PX. It associated operations are:

$$\begin{array}{ll} val_X : X \Rightarrow PX & 0_X : PX \\ let_{X,Y} : (X \Rightarrow PY) \times PX \Rightarrow PY & \cup_X : PX \times PX \Rightarrow PX \end{array}$$

and are subject to the following laws:

<u>P a commutative monad [23]:</u>

$$\left.\begin{array}{l} (a)\ let(f, val(x)) = f(x) \\ (b)\ let(\lambda x.\ val(x), p) = p \\ (c)\ let(\lambda x. let(\lambda y. h(x,y), q), p) = let(\lambda y. let(\lambda x. h(x,y), p), q) \end{array}\right\} \quad (12)$$

<u>$(PX, 0_X, \cup_X)$ a semilattice:</u>

$$\left.\begin{array}{ll} 0 \cup p = p = p \cup 0 & (p_1 \cup p_2) \cup p_3 = p_1 \cup (p_2 \cup p_3) \\ p \cup q = q \cup p & p \cup p = p \end{array}\right\} \quad (13)$$

<u>$\lambda x.\ let(f, x)$ a semilattice homomorphism:</u>

$$\left.\begin{array}{l} (a)\ let(f, 0) = 0 \\ (b)\ let(f, \cup(p,q)) = \cup(let(f,p), let(f,q)) \end{array}\right\} \quad (14)$$

In the domain-theoretic interpretation we also have the operation $\bot_X : PX$ such that

<u>$\lambda x.\ let(f, x)$ strict:</u>

$$let(f, \bot) = \bot.$$

- For a type X, we have the *dynamic-allocation type* δX. It associated operations are:

$$\mathsf{up}_X : X \Rightarrow \delta X, \quad \mathsf{swap}_X : \delta^2 X \Rightarrow \delta^2 X, \quad \delta_{X,Y} : (X \Rightarrow Y) \Rightarrow \delta X \Rightarrow \delta Y$$

and are subject to the following laws:

<u>Inherited properties from *swap* and *up* (see (11) above):</u>

$$\left.\begin{array}{l} (a)\ \mathsf{swap}_X \circ \mathsf{swap}_X = \mathrm{id}_{\delta^2 X} \\ (b)\ \delta(\mathsf{up}_X) \circ \mathsf{up}_X = \mathsf{up}_{\delta X} \circ \mathsf{up}_X \\ (c)\ \mathsf{swap}_X \circ \delta(\mathsf{up}_X) = \mathsf{up}_{\delta X} \end{array}\right\} \quad (15)$$

<u>δ a functor:</u>

$$\left.\begin{array}{l} (a)\ \delta_{X,X}(\mathrm{id}_X) = \mathrm{id}_{\delta X} \\ (b)\ (\delta_{Y,Z}\ g) \circ (\delta_{X,Y}\ f) = \delta_{X,Z}(g \circ f) \end{array}\right\} \quad (16)$$

<u>swap a natural transformation:</u>

$$(\delta^2{}_{X,Y}\ f) \circ \mathsf{swap}_X = \mathsf{swap}_Y \circ (\delta^2{}_{X,Y}\ f) \quad (17)$$

where in the last equation $\delta^2{}_{X,Y}$ stands for $\delta_{\delta X,\delta Y} \circ \delta_{X,Y}$.

In addition, we have the following type equalities:

$$\begin{array}{ll} \multicolumn{2}{l}{\underline{\delta \text{ preserves the unit, products, sums, and power types:}}} \\ \delta 1 = 1 & \delta(X \times Y) = \delta(X) \times \delta(Y) \\ \delta PX = P\delta X & \delta(X + Y) = \delta(X) + \delta(Y) \end{array}$$

accompanied by *preservation laws* among which the following will be needed later:

$$\left.\begin{array}{ll} \multicolumn{2}{l}{\underline{\delta \text{ preserves products, sums, and semilattice structure:}}} \\ (a)\ \delta\ \langle f, g\rangle = \langle \delta\ f, \delta\ g\rangle & (b)\ \delta\ \pi_i^{X,Y} = \pi_i^{\delta X,\delta Y}\ \ (i = 1,2) \\ (c)\ \delta\ \mathsf{in}_i^{X,Y} = \mathsf{in}_i^{\delta X,\delta Y}\ \ (i = 1,2) & (d)\ \delta\ [f, g] = [\delta\ f, \delta\ g] \\ (e)\ \delta 0_X = 0_{\delta X} & (f)\ \delta \cup_{PX} = \cup_{\delta PX} \end{array}\right\} \quad (18)$$

$$\mathsf{swap}_{X\times Y}(x, y) = (\mathsf{swap}_X x, \mathsf{swap}_Y y) \qquad (19)$$

and

$$\delta_{1,X} = \mathsf{up}_X. \qquad (20)$$

- We have a *type of names* N equipped with a decidable equality $\mathsf{eq} : N \times N \Rightarrow 2$ (here $2 = 1 + 1$) subject to the usual laws.

  The dynamic-allocation type over N satisfies the equality

  $$\delta(N) = N + 1 \qquad (21)$$

  with injections $\mathsf{up}_N : N \Rightarrow \delta N$ and $\mathsf{new} : \delta N$ (i.e. such that $\mathsf{in}_1^{N,1} = \mathsf{up}_N$ and $\mathsf{in}_2^{N,1} = \mathsf{new}$).

  A crucial law of swap is:

  $$\left.\begin{array}{l} \underline{\mathsf{swap} \text{ swaps } \delta\ \mathsf{new} \text{ and } \delta\ \mathsf{up}_N\ \mathsf{new}:} \\ \mathsf{swap}_N(\delta\ \mathsf{new}) = \delta\ \mathsf{up}_N\ \mathsf{new} \\ \mathsf{swap}_N(\delta\ \mathsf{up}_N\ \mathsf{new}) = \delta\ \mathsf{new} \end{array}\right\} \quad (22)$$

- We have a *type of agents* $A = PHA$ where $HX \overset{\text{def}}{=} N \times (N {\Rightarrow} X) + N \times N \times X + N \times \delta X + X$.

  We introduce some auxiliary notation (related to a suitable monad transformer —see [10]) which simplifies the description of the denotational semantics:

  - $\mathsf{S} : HA \Rightarrow A$ is given by $\mathsf{S}(x) = \mathit{val}(x)$, here we use that $A = PHA$.

    We write S_i $(i = 1, 4)$ for the i^{th} component of S; so that $\mathsf{S} = [\mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3, \mathsf{S}_4]$.
  - $\mathsf{C} : (HA{\Rightarrow}A){\times}A \Rightarrow A$ is given by $\mathsf{C}(f, p) = \mathit{let}(f, p)$, here we use that $A = PHA$.

    We write $\mathsf{CC} : (HA \times HA \Rightarrow A) \times A \times A \Rightarrow A$ for C iterated twice; that is, $\mathsf{CC}(h, p_1, p_2) = \mathsf{C}(\lambda x.\, \mathsf{C}(\lambda y.\, h(x, y), p_2), p_1)$.

S and C inherit the properties of *val* and *let* (see (12)–(14) above).

- In the domain-theoretic interpretation we have a *fixed-point operator*

$$\mathsf{fix}_{X \Rightarrow A} : ((X \Rightarrow A) \Rightarrow X \Rightarrow A) \Rightarrow X \Rightarrow A \tag{23}$$

satisfying the usual fixed-point property.

As a notational convention, for $f : X \Rightarrow A, x : X \vdash e[f, x] : A$, we write $F(x) = e[F, x]$ for $F = \mathsf{fix}(\lambda f.\ \lambda x.\ e[f, x])$.

We conclude the subsection with some sample derivations in the metalanguage.

EXAMPLE 4.1.

- Using (16b) and (18a) we may derive the following equation: for $a : X$, $f_i : X \Rightarrow Y_i$ $(i = 1, 2)$, $g : Y_1 \times Y_2 \Rightarrow Z$,

$$\delta\ (\lambda x : X.\, g(f_1 x, f_2 x))\ a \ = \ \delta\ g\ (\delta\ f_1\ a, \delta\ f_2\ a). \tag{24}$$

- To illustrate the use of the metalanguage we provide a derivation of the following: for $h : N \times N \Rightarrow X$,

$$\begin{aligned} &\mathsf{swap}_X(\delta\ (\lambda x : N.\, \delta\ (\lambda y : N.\, h(x, y))\ \mathsf{new})\ \mathsf{new}) \\ &\quad = \ \delta\ (\lambda y : N.\, \delta\ (\lambda x : N.\, h(x, y))\ \mathsf{new})\ \mathsf{new}. \end{aligned} \tag{25}$$

1. For $u : U$ and $d : \delta V$,

$$\begin{aligned} &\delta_{V,U}\ (\lambda y : V.\, u)\ d \\ &\quad = \ \delta_{V,U}\ ((\lambda x : 1.\ u) \circ (\lambda y : V.\ *))\ d \\ &\quad = \ \delta_{1,U}\ (\lambda x : 1.\ u)\ (\delta_{V,1}\ (\lambda y : V.\ *)\ d) \qquad \text{, by (16b)} \\ &\quad = \ \delta_{1,U}\ (\lambda x : 1.\ u)\ * \qquad \text{, using } \delta 1 = 1 \text{ and } X \Rightarrow 1 = 1 \\ &\quad = \ \mathsf{up}_U\ u \qquad \text{, using } 1 \Rightarrow X = X \text{ and (20)} \end{aligned} \tag{26}$$

2. For $h : N \times N \Rightarrow X$ and $x : N$,

$$\begin{aligned} &\delta\ (\lambda y : N.\ h(x, y))\ \mathsf{new} \\ &\quad = \ \delta\ (\lambda y : N.\ h((\lambda z : N.\ x)\ y), (\lambda z : N.\ z)\ y)\ \mathsf{new} \\ &\quad = \ \delta\ h\ (\delta\ (\lambda z : N.\ x)\ \mathsf{new}, \delta\ \mathrm{id}_N\ \mathsf{new}) \qquad \text{, by (24)} \\ &\quad = \ \delta\ h\ (\mathsf{up}_N\ x, \mathsf{new}) \qquad \text{, by (26) and (16a)} \end{aligned} \tag{27}$$

3. For $h : N \times N \Rightarrow X$,

$$\begin{aligned} &\delta\ (\lambda x : N.\ \delta\ (\lambda y : N.\ h(x, y))\ \mathsf{new})\ \mathsf{new} \\ &\quad = \ \delta\ (\lambda x : N.\ \delta\ h\ (\mathsf{up}_N\ x, \mathsf{new}))\ \mathsf{new} \qquad \text{, by (27)} \\ &\quad = \ \delta\ (\delta h)\ (\delta\ \mathsf{up}_N\ \mathsf{new}, \delta\ (\lambda z : N.\ \mathsf{new})\ \mathsf{new}) \qquad \text{, by (24)} \\ &\quad = \ \delta^2\ h\ (\delta\ \mathsf{up}_N\ \mathsf{new}, \delta\ \mathsf{new}) \qquad \text{, by (26)} \end{aligned} \tag{28}$$

4. For $h : N \times N \Rightarrow X$,

$$\begin{aligned}
&\mathsf{swap}_X(\delta\ (\lambda x : N.\,\delta\ (\lambda y : N.\,h(x,y))\ \mathsf{new})\ \mathsf{new})\\
&\quad = \mathsf{swap}_X(\delta^2\ h\ (\delta\ \mathsf{up}_N\ \mathsf{new}, \delta\ \mathsf{new})) \qquad \text{, by (28)}\\
&\quad = \delta^2\ h\ (\mathsf{swap}_{N\times N}(\delta\ \mathsf{up}_N\ \mathsf{new}, \delta\ \mathsf{new})) \qquad \text{, by (17)}\\
&\quad = \delta^2\ h\ (\mathsf{swap}_N(\delta\ \mathsf{up}_N\ \mathsf{new}), \mathsf{swap}_N(\delta\ \mathsf{new})) \qquad \text{, by (19)}\\
&\quad = \delta^2\ h\ (\delta\ \mathsf{new}, \delta\ \mathsf{up}_N\ \mathsf{new}) \qquad \text{, by (22) and (15a)}\\
&\quad = \delta\ (\lambda y : N.\,\delta\ (\lambda x : N.\,h(x,y))\ \mathsf{new})\ \mathsf{new}
\end{aligned}$$

where the last equality follows from calculations analogous to those leading to (28).

- By (12c) we have:

$$\mathsf{CC}(K, p_1, p_2) \;=\; \mathsf{CC}(K \circ \langle \pi_2, \pi_1\rangle, p_2, p_1) \tag{29}$$

4.6. Interpretation

We give the interpretation of the term constructors for an *abstract syntax* of the π-calculus (c.f. Definition 1.1). The interpretation is defined by translation into the metalanguage of the previous subsection as follows:

$\mathsf{nil} : A$	deadlock
$\mathsf{sum} : A, A \to A$	non-deterministic choice

are given by the semilattice structure of PHA.

$\mathsf{M} : N, N, A \to A$	matching
$\mathsf{MM} : N, N, A \to A$	mismatching

are defined by case analysis using the decidable equality on the type of names.

$\mathsf{in} : N, (N \Rightarrow A) \to A$	input
$\mathsf{out} : N, N, A \to A$	output
$\mathsf{bout} : N, (N \Rightarrow A) \to A$	bound output
$\mathsf{tau} : A \to A$	internal action

correspond, except for bout, to components of the operation $\mathsf{S} : HA \Rightarrow A$, that is $[\mathsf{in}, \mathsf{out}, \mathsf{S}_3, \mathsf{tau}] = \mathsf{S}$. Bounded output is given by $\mathsf{bout}(a, f) = \mathsf{S}_3(a, \delta\ f\ \mathsf{new})$ —see Proposition 4.2 and the remark after it.

$$\mathsf{R}(p) \overset{\text{def}}{=} let(\begin{bmatrix} \lambda a : \delta N, f : \delta(N \Rightarrow A).\begin{bmatrix} \lambda a' : N.\,\mathsf{S}_1(a', \lambda b : N.\,\mathsf{R}(\delta\ (\lambda f' : N \Rightarrow A.\, f'b)\ f)) \\ \lambda t : 1.\,0 \end{bmatrix} a \\ \lambda a : \delta N, b : \delta N, p' : \delta A.\begin{bmatrix} \lambda a' : N.\begin{bmatrix} \lambda b' : N.\,\mathsf{S}_2(a', b', \mathsf{R}\ p') \\ \lambda t : 1.\,\mathsf{S}_3(a', p') \end{bmatrix} b \\ \lambda t : 1.\,0 \end{bmatrix} a \\ \lambda a : \delta N, p' : \delta^2 A.\begin{bmatrix} \lambda a' : N.\,\mathsf{S}_3(a', \delta\ \mathsf{R}\ (\mathsf{swap}_A\ p')) \\ \lambda t : 1.\,0 \end{bmatrix} a \\ \lambda p' : \delta A.\,\mathsf{S}_4(\mathsf{R}\ p') \end{bmatrix}, p)$$

TABLE 4

$\mathsf{R} : \boldsymbol{\delta A \Rightarrow A}$

$\mathsf{res} : (N \Rightarrow A) \to A$	local name

is given by $\mathsf{res}(f) = \mathsf{R}(\delta\ f\ \mathsf{new})$ where the operation $\mathsf{R} : \delta A \Rightarrow A$, maps, at stage n, a process with $n+1$ free names to one with n free names by *restricting* on the last allocated name. The definition of R, given in Table 4, relies on the type equalities

$$\delta A = P(\delta N \times \delta(N \Rightarrow A) + \delta N \times \delta N \times \delta A + \delta N \times \delta^2 A + \delta A) \quad \text{and} \quad \delta N = N + 1$$

and inspects whether each action capability involves the last allocated name (i.e. the new one) or not:

- In case of an input at a: if a is new, then we cancel the input capability, otherwise we allow the input and restrict on the continuation.
- In case of a free output at a of b: if a is new, then we cancel the output capability; if a is not new but b is, then we make the free output into a bound output; if neither a nor b are new, then we allow the output and restrict on the continuation.
- In case of a bound output at a: if a is new, then we cancel the output capability, otherwise we allow the output and restrict on the second last allocated name (*not* on the last allocated one as this is the one being allocated by the bound output).
- In case of an internal action: we allow the action and restrict on the continuation.

Note that the definition of res in terms of R mimics that of bout in terms of S_3.

$\mathsf{par} : A, A \to A$	parallel composition
$\mathsf{lm} : A, A \to A$	left merge
$\mathsf{syn} : A, A \to A$	synchronisation

are defined by mutual recursion as follows:

$\mathsf{par}(p, q) = \mathsf{sum}(\mathsf{lm}(p, q), \mathsf{lm}(q, p), \mathsf{syn}(p, q), \mathsf{syn}(q, p))$.

$$\left[\begin{array}{r|cccc}
 & x_2 : N, f_2 : N \Rightarrow A & x_2 : N, y_2 : N, q_2 : A & x_2 : N, q_2' : \delta A & q_2 : A \\
\hline
x_1 : N, f_1 : N \Rightarrow A & \mathsf{nil} & \mathsf{com}(x_1, x_2, f_1(y_2), q_2) & \mathsf{close}(x_1, x_2, (\delta\ f_1\ \mathsf{new}), q_2') & \mathsf{nil} \\
x_1 : N, y_1 : N, q_1 : A & \mathsf{com}(x_2, x_1, f_2(y_1), q_1) & \mathsf{nil} & \mathsf{nil} & \mathsf{nil} \\
x_1 : N, q_1' : \delta A & \mathsf{close}(x_2, x_1, (\delta\ f_2\ \mathsf{new}), q_1') & \mathsf{nil} & \mathsf{nil} & \mathsf{nil} \\
q_1 : A & \mathsf{nil} & \mathsf{nil} & \mathsf{nil} & \mathsf{nil}
\end{array}\right]$$

TABLE 5
$\boldsymbol{K : (HA \times HA) \Rightarrow A}$

$$\mathsf{lm}(p, q) = \mathsf{C}(\left[\begin{array}{l}
\lambda a : N, f : N \Rightarrow A.\,\mathsf{in}(a, \lambda b : N.\,\mathsf{par}(fb, q)) \\
\lambda a : N, b : N, p' : A.\,\mathsf{out}(a, b, \mathsf{par}(p', q)) \\
\lambda a : N, p' : \delta A.\,\mathsf{S}_3(a, \delta\ \mathsf{par}\ (p', \mathsf{up}\ q)) \\
\lambda p' : A.\,\mathsf{tau}(\mathsf{par}(p', q))
\end{array}\right], p).$$

When two processes try to communicate there are 16 possibilities to consider, since each process may perform 4 possible actions. This explains the case analysis in the following definition:

$\mathsf{syn}(p_1, p_2) = \mathsf{CC}(K, p_1, p_2)$

where the cases covered by $K : (HA \times HA) \Rightarrow A$ are summarised in the *symmetric* Table 5 in which $\mathsf{com} : N \times N \times A \times A \Rightarrow A$ and $\mathsf{close} : N \times N \times \delta A \times \delta A \Rightarrow A$ are defined as $\mathsf{com}(x, y, p, q) = \mathsf{M}(x, y, \mathsf{tau}(\mathsf{par}(p, q)))$ and $\mathsf{close}(x, y, p', q') = \mathsf{M}(x, y, \mathsf{tau}(\mathsf{R}(\delta\ \mathsf{par}\ (p', q'))))$.

$\mathsf{!in} : N, (N \Rightarrow A) \to A$	replicated input
$\mathsf{!out} : N, N, A \to A$	replicated output
$\mathsf{!bout} : N, (N \Rightarrow A) \to A$	replicated bound output
$\mathsf{!tau} : A \to A$	replicated internal action

are defined by recursion (but not mutual recursion) as follows:

$\mathsf{!in}(a, f) = \mathsf{in}(a, \lambda b : N.\,\mathsf{par}(fb, \mathsf{!in}(a, f)))$.

$\mathsf{!out}(a, b, p) = \mathsf{out}(a, b, \mathsf{par}(p, \mathsf{!out}(a, b, p)))$.

$\mathsf{!bout}(a, f) = \mathsf{bout}(a, \lambda b : N.\,\mathsf{par}(fb, \mathsf{!bout}(a, f)))$.

$\mathsf{!tau}(p) = \mathsf{tau}(\mathsf{par}(p, \mathsf{!tau}(p)))$.

Open interpretation. Given the interpretation/translation of the above term constructors it is straightforward to extend it to names and well-formed processes following the paradigm of categorical logic [20]. For a name $a \in V$, $\mathcal{O}[\![V \vdash a]\!]$ is the obvious projection $N^{|V|} \dot{\to} N$ in $\mathcal{C}^{\mathcal{I}}$. For a process $P \in \mathcal{P}r_V$, $\mathcal{O}[\![V \vdash P]\!]$ is a morphism $N^{|V|} \dot{\to} A$ in $\mathcal{C}^{\mathcal{I}}$ defined by induction on the structure of P (see Subsection A.1). For example,

$$\mathcal{O}[\![V \vdash P_1 \mid P_2]\!] \overset{\text{def}}{=} \mathsf{par} \circ \langle \mathcal{O}[\![V \vdash P_1]\!], \mathcal{O}[\![V \vdash P_2]\!] \rangle$$

and

$$\mathcal{O}[\![V \vdash a(b).\,P]\!] \overset{\text{def}}{=} \mathsf{in} \circ \langle \mathcal{O}[\![V \vdash a]\!], \lambda\, \mathcal{O}[\![V, b \vdash P]\!]\rangle.$$

Closed interpretation. Bisimulation cannot correspond to the above interpretation, because the operational semantics considers free names of a process semantically different. In the denotational semantics this requirement can be captured smoothly by exploiting the functor category structure and adopting a *closed* interpretation such that for V with $\mid V \mid = n$, $\mathcal{C}[\![V \vdash a]\!] \in N(n) = n$ and $\mathcal{C}[\![V \vdash P]\!] \in A(n)$. Then the closed interpretation is defined in terms of the *open* interpretation: $\mathcal{C}[\![V \vdash \_\,]\!] \overset{\text{def}}{=} \mathcal{O}[\![V \vdash \_\,]\!]_n(0, \ldots, n-1)$.

It follows that the close interpretation is *compositional* on all constructs but input; the clause for input cannot be purely compositional because late bisimulation is not preserved by this operator.

Lemma 4.1.

1.$\mathcal{C}[\![V \vdash \overline{a}b.\,P]\!] = \mathsf{out}_{|V|}(\mathcal{C}[\![V \vdash a]\!], \mathcal{C}[\![V \vdash b]\!], \mathcal{C}[\![V \vdash P]\!])$.
2.$\mathcal{C}[\![V \vdash \overline{a}(b).\,P]\!] = (\mathsf{S}_3)_{|V|}(\mathcal{C}[\![V \vdash a]\!], \mathcal{C}[\![V, b \vdash P]\!])$.
3.*For* $V \equiv a_0, \ldots, a_{n-1}$,
$\mathcal{C}[\![V \vdash a(b).\,P]\!] = \mathsf{in}_n(\mathcal{C}[\![V \vdash a]\!], \langle \mathcal{C}[\![V \vdash P\{{}^{a_i}\!/\!_b\}]\!]\rangle_{i \in n}, \mathcal{C}[\![V, b \vdash P]\!])$.
4.$\mathcal{C}[\![V \vdash \tau.\,P]\!] = \mathsf{tau}_{|V|}(\mathcal{C}[\![V \vdash P]\!])$.
5.$\mathcal{C}[\![V \vdash \mathbf{0}]\!] = \mathsf{nil}_{|V|}$.
6.$\mathcal{C}[\![V \vdash P + Q]\!] = \mathsf{sum}_{|V|}(\mathcal{C}[\![V \vdash P]\!], \mathcal{C}[\![V \vdash Q]\!])$.
7.$\mathcal{C}[\![V \vdash P \mid Q]\!] = \mathsf{par}_{|V|}(\mathcal{C}[\![V \vdash P]\!], \mathcal{C}[\![V \vdash Q]\!])$.
8.$\mathcal{C}[\![V \vdash \boldsymbol{\nu} a\,.\,P]\!] = \mathsf{R}_{|V|}(\mathcal{C}[\![V, a \vdash P]\!])$.
9.$\mathcal{C}[\![V \vdash P \lfloor\!\lfloor Q]\!] = \mathsf{lm}_{|V|}(\mathcal{C}[\![V \vdash P]\!], \mathcal{C}[\![V \vdash Q]\!])$.
10.$\mathcal{C}[\![V \vdash P \mid\mid Q]\!] = \mathsf{syn}_{|V|}(\mathcal{C}[\![V \vdash P]\!], \mathcal{C}[\![V \vdash Q]\!])$.
11.$\mathcal{C}[\![V \vdash [a = b]P]\!] = \mathsf{M}_{|V|}(\mathcal{C}[\![V \vdash a]\!], \mathcal{C}[\![V \vdash b]\!], \mathcal{C}[\![V \vdash P]\!])$.
12.$\mathcal{O}[\![V \vdash [a \neq b]P]\!] = \mathsf{MM}_{|V|}(\mathcal{C}[\![V \vdash a]\!], \mathcal{C}[\![V \vdash b]\!], \mathcal{C}[\![V \vdash P]\!])$.

Proof. See Subsection A.2. ■

The induced semantics is preserved by renaming (see Lemma A.1); that is, for $b \notin V, W$,

$$\mathcal{C}[\![V, a, W \vdash P]\!] = \mathcal{C}[\![V, b, W \vdash P\{{}^{b}\!/\!_{a}\}]\!] \tag{30}$$

5. DENOTATIONAL VALIDITY

We take a first step in relating the operational and the denotational semantics of the π-calculus. In particular, we show that the closed interpretation validates the equations of the system $\mathcal{A}_\pi$ (Theorem 5.1) and the ω-birule (1) (Theorem 5.2). These results will be important for establishing the full abstraction of the denotational semantics.

5.1. Denotational validity of the equations of the system $\mathcal{A}_\pi$

We show that the closed interpretation of the π-calculus validates both the axioms and inference rules of $\mathcal{A}_\pi$. Roughly, we proceed as follows:

1. we translate the axioms into the metalanguage and establish their validity using the equational theory of the metalanguage; and
2. we establish the validity of the inference rules from the *quasi-compositionality* (Lemma 4.1) of the closed interpretation.

LEMMA 5.1. *In the domain-theoretic interpretation, the denotational semantics validates the following equalities (which are semantic counterparts of the axioms in Table 2 except for axioms* C2 *and* C4*):*

`Summation` $p, q, r : A$

S1	$\mathsf{sum}(p, \mathsf{nil})$	$=$	p
S2	$\mathsf{sum}(p, q)$	$=$	$\mathsf{sum}(q, p)$
S3	$\mathsf{sum}(p, \mathsf{sum}(q, r))$	$=$	$\mathsf{sum}(\mathsf{sum}(p, q), r)$
S4	$\mathsf{sum}(p, p)$	$=$	p

`Restriction` $a, b : N \quad f, g : N \Rightarrow A \quad h : N \times N \Rightarrow A$

R1	$\mathsf{res}(\lambda x : N.\, \mathsf{sum}(fx, gx))$	$=$	$\mathsf{sum}(\mathsf{res}\, f, \mathsf{res}\, g)$
R2-in	$\mathsf{res}(\lambda x : N.\, \mathsf{in}(a, \lambda y : N.\, h(x, y)))$	$=$	$\mathsf{in}(a, \lambda y : N.\, \mathsf{res}(\lambda x : N.\, h(x, y)))$
R2-out	$\mathsf{res}(\lambda x : N.\, \mathsf{out}(a, b, fx))$	$=$	$\mathsf{out}(a, b, \mathsf{res}\, f)$
R2-bout	$\mathsf{res}(\lambda x : N.\, \mathsf{bout}(a, \lambda y : N.\, h(x, y)))$	$=$	$\mathsf{bout}(a, \lambda y : N.\, \mathsf{res}(\lambda x : N.\, h(x, y)))$
R2-tau	$\mathsf{res}(\lambda x : N.\, \mathsf{tau}(fx))$	$=$	$\mathsf{tau}(\mathsf{res}\, f)$
R3-in	$\mathsf{res}(\lambda x : N.\, \mathsf{in}(x, hx))$	$=$	nil
R3-out	$\mathsf{res}(\lambda x : N.\, \mathsf{out}(x, y, fx))$	$=$	nil
R3-bout	$\mathsf{res}(\lambda x : N.\, \mathsf{bout}(x, \lambda y : N.\, h(x, y)))$	$=$	nil
R4	$\mathsf{res}(\lambda x : N.\, \mathsf{out}(a, x, fx))$	$=$	$\mathsf{bout}(a, f)$
R5	$\mathsf{res}(\lambda x : N.\, \mathsf{nil})$	$=$	nil

`Conditionals` $a, b : N \quad p : A$

C1	$\mathsf{M}(a, a, p)$	$=$	p
C3	$\mathsf{MM}(a, a, p)$	$=$	nil

`Replication` $a, b : N \quad p : A \quad f : N \Rightarrow A$

Rep-in	$!\mathsf{in}(a, f)$	$=$	$\mathsf{in}(a, \lambda x : N.\, \mathsf{par}(fx, !\mathsf{in}(a, f)))$
Rep-out	$!\mathsf{out}(a, b, p)$	$=$	$\mathsf{out}(a, b, \mathsf{par}(p, !\mathsf{out}(a, b, p)))$
Rep-bout	$!\mathsf{bout}(a, f)$	$=$	$\mathsf{bout}(a, \lambda x : N.\, \mathsf{par}(fx, !\mathsf{bout}(a, f)))$
Rep-tau	$!\mathsf{tau}(p)$	$=$	$\mathsf{tau}(\mathsf{par}(p, !\mathsf{tau}\, p))$

`Parallel` $p, q : A$

`Par` $\mathsf{par}(p, q) \;=\; \mathsf{sum}(\mathsf{sum}(\mathsf{lm}(p, q), \mathsf{lm}(q, p)), \mathsf{syn}(p, q))$

`Left Merge` $a, b : N \quad p, q, r : A \quad f : N \Rightarrow A$

LM1	$\mathsf{lm}(\mathsf{nil}, p)$	$=$	nil
LM2	$\mathsf{lm}(\mathsf{sum}(p, q), r)$	$=$	$\mathsf{sum}(\mathsf{lm}(p, r), \mathsf{lm}(q, r))$
LM3-in	$\mathsf{lm}(\mathsf{in}(a, f), p)$	$=$	$\mathsf{in}(a, \lambda x : N.\, \mathsf{par}(fx, p))$
LM3-out	$\mathsf{lm}(\mathsf{out}(a, b, p), q)$	$=$	$\mathsf{out}(a, b, \mathsf{par}(p, q))$
LM3-bout	$\mathsf{lm}(\mathsf{bout}(a, f), p)$	$=$	$\mathsf{bout}(a, \lambda x : N.\, \mathsf{par}(fx, p))$
LM3-tau	$\mathsf{lm}(\mathsf{tau}\, p, q)$	$=$	$\mathsf{tau}(\mathsf{par}(p, q))$

`Synchronisation`	$a,b,c,d:N \quad p,q,r:A \quad f,g:N\Rightarrow A$
`Syn1`	$\mathsf{syn}(\mathsf{nil},p) = \mathsf{nil}$
`Syn2`	$\mathsf{syn}(p,q) = \mathsf{syn}(q,p)$
`Syn3`	$\mathsf{syn}(\mathsf{sum}(p,q),r) = \mathsf{sum}(\mathsf{syn}(p,r),\mathsf{syn}(q,r))$
`Syn4`	$\mathsf{syn}(\mathsf{in}(a,f),\mathsf{out}(b,c,q)) = \mathsf{M}(a,b,\mathsf{tau}(\mathsf{par}(fc,q)))$
`Syn5`	$\mathsf{syn}(\mathsf{in}(a,f),\mathsf{bout}(b,g)) = \mathsf{M}(a,b,\mathsf{tau}(\mathsf{res}(\lambda x:N.\,\mathsf{par}(fx,gx))))$
`Syn6`	$\mathsf{syn}(\mathsf{tau}\,p,q) = \mathsf{nil}$
`Syn7`	$\mathsf{syn}(\mathsf{in}(a,f),\mathsf{in}(b,g)) = \mathsf{nil}$
`Syn8-out/out`	$\mathsf{syn}(\mathsf{out}(a,b,p),\mathsf{out}(c,d,q)) = \mathsf{nil}$
`Syn8-out/bout`	$\mathsf{syn}(\mathsf{out}(a,b,p),\mathsf{bout}(c,f)) = \mathsf{nil}$
`Syn8-bout/bout`	$\mathsf{syn}(\mathsf{bout}(a,f),\mathsf{bout}(b,g)) = \mathsf{nil}$

Remark. Notice how in translating the laws into the metalanguage π-bindings become λ-bindings; whilst substitutions become applications.

Proof. We only consider the interesting cases:

(`R1`) $\mathsf{res}(\lambda x.\,\mathsf{sum}(fx,gx))$

$$\begin{aligned}
&= \mathsf{R}(\delta\ (\lambda x.\,\cup_{PHA}(fx,gx))\ \mathsf{new}) && \text{, by definition of } \mathsf{R}\\
&= \mathsf{R}(\delta\cup_{PHA}(\delta\ f\ \mathsf{new},\delta\ g\ \mathsf{new})) && \text{, by (24)}\\
&= \mathsf{R}(\cup_{P\delta HA}(\delta\ f\ \mathsf{new},\delta\ g\ \mathsf{new})) && \text{, by (18f)}\\
&= \cup_{PHA}(\mathsf{R}(\delta\ f\ \mathsf{new}),\mathsf{R}(\delta\ g\ \mathsf{new})) && \text{, by (16b)}\\
&= \mathsf{sum}(\mathsf{res}\ f,\mathsf{res}\ g).
\end{aligned}$$

(`R2-in`) $\mathsf{res}(\lambda x.\,\mathsf{in}(a,\lambda y.\,h(x,y)))$

$$\begin{aligned}
&= \mathsf{R}(\delta\ (\lambda x.\,\mathsf{S}_1(a,\lambda y.\,h(x,y)))\ \mathsf{new}) && \text{, by definition of } \mathsf{R}\\
&= \mathsf{R}[\delta\mathsf{S}_1(\delta\ a,\delta\ (\lambda x.\,\lambda y.\,h(x,y))\ \mathsf{new})] && \text{, by (24)}\\
&= \mathsf{S}_1(a,\lambda b.\,\mathsf{R}(\delta\ (\lambda f'.\,f'b)\ (\delta\ (\lambda x.\,\lambda y.\,h(x,y))\ \mathsf{new}))) && \text{, by definition of } \mathsf{R}\text{,}\\
& && \text{(12a), and (18c)}\\
&= \mathsf{S}_1(a,\lambda b.\,\mathsf{R}(\delta\ (\lambda x.\,h(x,b))\ \mathsf{new})) && \text{, by (16b)}\\
&= \mathsf{in}(a,\lambda b.\,\mathsf{res}(\lambda x.\,h(x,b))).
\end{aligned}$$

(`R2-bout`) $\mathsf{res}(\lambda x.\,\mathsf{bout}(a,\lambda y.\,h(x,y)))$

$$\begin{aligned}
&= \mathsf{R}(\delta\ (\lambda x.\,\mathsf{bout}(a,\lambda y.\,h(x,y)))\ \mathsf{new}) && \text{, by def. of } \mathsf{R}\\
&= \mathsf{R}[\delta\ (\lambda x.\,\mathsf{S}_3(a,\delta\ (\lambda y.\,h(x,y))\ \mathsf{new}))\ \mathsf{new}] && \text{, by def. of } \mathsf{bout}\\
&= \mathsf{R}[\delta\mathsf{S}_3(\delta\ a,\delta\ (\lambda x.\,\delta\ (\lambda y.\,h(x,y))\ \mathsf{new})\ \mathsf{new})] && \text{, by (24)}\\
&= \mathsf{S}_3(a,\delta\ \mathsf{R}\ (\mathsf{swap}(\delta\ (\lambda x.\,\delta\ (\lambda y.\,h(x,y))\ \mathsf{new})\ \mathsf{new}))) && \text{, by def. of } \mathsf{R}\text{,}\\
& && \text{(12a), and (18c)}\\
&= \mathsf{S}_3(a,\delta\ \mathsf{R}\ (\delta\ (\lambda y.\,\delta\ (\lambda x.\,h(x,y))\ \mathsf{new})\ \mathsf{new})) && \text{, by (25)}\\
&= \mathsf{S}_3(a,\delta\ (\lambda y.\,\mathsf{R}(\delta\ (\lambda x.\,h(x,y))\ \mathsf{new}))\ \mathsf{new}) && \text{, by (24)}\\
&= \mathsf{S}_3(a,\delta\ (\lambda y.\,\mathsf{res}(\lambda x.\,h(x,y)))\ \mathsf{new}) && \text{, by def. of } \mathsf{R}\\
&= \mathsf{bout}(a,\lambda y.\,\mathsf{res}(\lambda x.\,h(x,y))) && \text{, by def. of } \mathsf{bout}
\end{aligned}$$

(LM3-bout) $\mathsf{lm}(\mathsf{bout}(a,f),p)$

$$\begin{aligned}
&= \mathsf{lm}(\mathsf{S}_3(a,\delta\ f\ \mathsf{new}),p) && \text{, by definition of } \mathsf{bout}\\
&= \mathsf{S}_3(a,\delta\ \mathsf{par}\ (\delta\ f\ \mathsf{new},\mathsf{up}\ p)) && \text{, by definition of } \mathsf{lm}\\
&= \mathsf{S}_3(a,\delta\ \mathsf{par}\ (\delta\ f\ \mathsf{new},\delta\ p)) && \text{, by (20)}\\
&= \mathsf{S}_3(a,\delta\ (\lambda x.\,\mathsf{par}(fx,p))\ \mathsf{new}) && \text{, by (24)}\\
&= \mathsf{bout}(a,\lambda x.\,(fx,p)) && \text{, by definition of } \mathsf{bout}
\end{aligned}$$

(Syn2)
$$\begin{aligned}
\mathsf{syn}(p,q) &= \mathsf{CC}(K,(p,q)) && \text{, by definition of } \mathsf{syn}\\
&= \mathsf{CC}(K\circ\langle\pi_2,\pi_1\rangle,(q,p)) && \text{, because } K = K\circ\langle\pi_2,\pi_1\rangle\\
&= \mathsf{CC}(K,(q,p)) && \text{, by (29)}\\
&= \mathsf{syn}(q,p) && \text{, by definition of } \mathsf{syn}
\end{aligned}$$

(Syn5) $\mathsf{syn}(\mathsf{in}(a,f),\mathsf{bout}(b,g))$

$$\begin{aligned}
&= \mathsf{syn}(\mathsf{S}_1(a,f),\mathsf{S}_3(b,\delta\ g\ \mathsf{new})) && \text{, by definition of } \mathsf{in} \text{ and } \mathsf{bout}\\
&= \mathsf{close}(a,b,\delta\ f\ \mathsf{new},\delta\ g\ \mathsf{new}) && \text{, by definition of } \mathsf{syn}\\
&= \mathsf{M}(a,b,\mathsf{tau}(\mathsf{R}(\delta\ \mathsf{par}\ (\delta\ f\ \mathsf{new},\delta\ g\ \mathsf{new})))) && \text{, by definition of } \mathsf{close}\\
&= \mathsf{M}(a,b,\mathsf{tau}(\mathsf{R}(\delta\ (\lambda x.\,\mathsf{par}(fx,gx))\ \mathsf{new}))) && \text{, by (24)}\\
&= \mathsf{M}(a,b,\mathsf{tau}(\mathsf{res}(\lambda x.\,\mathsf{par}(fx,gx)))). && \text{, by definition of } \mathsf{res}
\end{aligned}$$
■

COROLLARY 5.1. *In the domain-theoretic interpretation, for every $V\vdash P$ and $V\vdash Q$ such that $P=Q$ is an axiom of $\mathcal{A}_\pi$ (see Table 2), we have that $\mathcal{C}[\![V\vdash P]\!]=\mathcal{C}[\![V\vdash Q]\!]$.*

Proof. We only consider the axioms C2 and C4; the validity of the rest of the axioms follows from Lemma 5.1 using Lemma A.1.

(C2/C4) For $n\geq 1$, let $V\equiv a_0,\ldots,a_{n-1}$ and let $0\leq i,j\leq n-1$.
We have that

$$\begin{aligned}
\mathcal{O}[\![V\vdash[a_i=a_j]P]\!] &= \mathsf{M}\circ\langle\mathcal{O}[\![V\vdash a_i]\!],\mathcal{O}[\![V\vdash a_j]\!],\mathcal{O}[\![V\vdash P]\!]\rangle\\
&= \lambda v.\left[\begin{array}{l}\lambda t:1.\,\mathcal{O}[\![V\vdash P]\!]\\ \lambda t:1.\,\mathsf{nil}\end{array}\right](\mathsf{eq}(\pi_i v,\pi_j v)).
\end{aligned}$$

Then, for $0\leq k_l\leq n-1$ $(0\leq l\leq n-1)$ we have that

$$\mathcal{O}[\![V\vdash[a_i=a_j]P]\!]_n(k_0,\ldots,k_{n-1}) = \begin{cases}\mathcal{O}[\![V\vdash P]\!]_n(k_0,\ldots,k_{n-1}) & \text{, if } k_i=k_j.\\ \mathsf{nil}_n & \text{, if } k_i\neq k_j.\end{cases}$$

Thus,

$$\mathcal{C}[\![V\vdash[a_i=a_j]P]\!] = \begin{cases}\mathcal{C}[\![V\vdash P]\!] & \text{, if } i=j.\\ \mathcal{C}[\![V\vdash \mathbf{0}]\!] & \text{, if } i\neq j.\end{cases}$$

■

Finally, we can establish the denotational validity of the equations of the system $\mathcal{A}_\pi$.

THEOREM 5.1. *In the domain-theoretic interpretation, for $V \vdash P$ and $V \vdash Q$, if $\mathcal{A}_\pi \models P = Q$ then $\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash Q]\!]$.*

Proof. Because the closed interpretation, in addition to validating the axioms of $\mathcal{A}_\pi$ (see Corollary 5.1), validates the inference rules of $\mathcal{A}_\pi$ (see Table 3); essentially as a consequence of Lemma 4.1 using (30). ■

From the definition of Exp_V^n (see (6) in Subsection 2.4) it follows that the closed domain-theoretic interpretation of a process equals that of its expansions.

COROLLARY 5.2. *In the domain-theoretic interpretation, for $P \in \mathcal{P}r_V$, we have that $\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash Exp_V^n(P)]\!]$ for all n.*

5.2. Denotational validity of the ω-birule

We show that the closed domain-theoretic interpretation validates the ω-birule (1).

We start by considering the *iterator* $\Phi : (A \Rightarrow A) \Rightarrow A \Rightarrow A$ defined, up to the equality $A = PHA$, as $\lambda h.\ P(Hh)$ where

$$P_{X,Y} \stackrel{\text{def}}{=} \lambda f.\ \lambda p.\ let(val \circ f, p) : (X \Rightarrow Y) \Rightarrow PX \Rightarrow PY$$

is the internalisation of the action of P on morphisms. It follows, by the initiality property of $A \stackrel{\text{def}}{=} \mu X.\ PHX$, that Φ has a unique fixed-point; viz. id_A.

Next we introduce semantic counterparts of the syntactic functions Nil_V^n and Bot_V^n of Subsection 2.4; by abuse of notation they will be denoted in the same way. The semantic functions $\mathrm{Nil}^n : A \Rightarrow A$ and $\mathrm{Bot}^n : A \Rightarrow A$ are respectively defined by iterating Φ n times at $\lambda p : A.\ 0$ and $\lambda p : A.\ \bot$. Formally, we set $\mathrm{Nil}^0 \stackrel{\text{def}}{=} \lambda p.\ \mathbf{0}$; $\mathrm{Bot}^0 \stackrel{\text{def}}{=} \lambda p.\ \bot$; and, for $f \in \{\mathrm{Nil}, \mathrm{Bot}\}$, $f^{n+1} \stackrel{\text{def}}{=} \Phi(f^n)$.

It follows by induction, using (12) and (14), that the semantic functions *mimic* the syntactic ones.

LEMMA 5.2. *Let $f, g \in \{\mathrm{Nil}, \mathrm{Bot}\}$.*

1.For $P \in \mathcal{E}_\bot\mathcal{X}_V^n$, $f_{|V|}^n(\mathcal{C}[\![V \vdash P]\!]) = \mathcal{C}[\![V \vdash f_V^n(P)]\!]$.
2.For $p \in A_l$, $f_l^n(g_l^{n+m}(p)) = f_l^n(p)$.

COROLLARY 5.3. *Let $P, Q \in \mathcal{E}_\bot\mathcal{X}_V^n$. For all n,*

$$\mathcal{C}[\![V \vdash \mathrm{Nil}_V^n(P)]\!] = \mathcal{C}[\![V \vdash \mathrm{Nil}_V^n(Q)]\!] \quad \textit{iff} \quad \mathrm{Bot}_V^n(\mathcal{C}[\![V \vdash P]\!]) = \mathrm{Bot}_V^n(\mathcal{C}[\![V \vdash Q]\!])$$

Proof. For the left to right direction we have that by hypothesis and by Lemma 5.2 (1),

$$\mathrm{Nil}_V^n(\mathcal{C}[\![V \vdash P]\!]) \ = \ \mathrm{Nil}_V^n(\mathcal{C}[\![V \vdash Q]\!]) \qquad (31)$$

Then,

$$\begin{aligned}
\mathrm{Bot}^n_V(\mathcal{C}[\![V \vdash P]\!]) &= \mathrm{Bot}^n_V(\mathrm{Nil}^n_V(\mathcal{C}[\![V \vdash P]\!])) &&\text{, by Lemma 5.2 (2)}\\
&= \mathrm{Bot}^n_V(\mathrm{Nil}^n_V(\mathcal{C}[\![V \vdash Q]\!])) &&\text{, using (31)}\\
&= \mathrm{Bot}^n_V(\mathcal{C}[\![V \vdash Q]\!]) &&\text{, by Lemma 5.2 (2)}
\end{aligned}$$

The right to left direction is proved by a symmetric argument. ■

We can finally establish the validity of the ω-birule in the close domain-theoretic interpretation.

THEOREM 5.2 (Denotational validity of the ω-birule). *For $P, Q \in \mathcal{P}r_V$, the following are equivalent:*

1.$\mathcal{C}[\![V \vdash \mathrm{Nil}^n_V(\mathit{Exp}^n_V(P))]\!] = \mathcal{C}[\![V \vdash \mathrm{Nil}^n_V(\mathit{Exp}^n_V(Q))]\!]$, *for all n.*
2.$\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash Q]\!]$.

Proof. We have the following sequence of equivalences:

$$\begin{aligned}
&\forall n.\ \mathcal{C}[\![V \vdash \mathrm{Nil}^n_V(\mathrm{Exp}^n_V(P))]\!] = \mathcal{C}[\![V \vdash \mathrm{Nil}^n_V(\mathrm{Exp}^n_V(Q))]\!]\\
&\quad\text{iff } \forall n.\ \mathrm{Bot}^n_V(\mathcal{C}[\![V \vdash \mathrm{Exp}^n_V(P)]\!]) = \mathrm{Bot}^n_V(\mathcal{C}[\![V \vdash \mathrm{Exp}^n_V(Q)]\!])\\
&\qquad\qquad\text{, by Corollary 5.3}\\
&\quad\text{iff } \forall n.\ \mathrm{Bot}^n_V(\mathcal{C}[\![V \vdash P]\!]) = \mathrm{Bot}^n_V(\mathcal{C}[\![V \vdash Q]\!])\\
&\qquad\qquad\text{, by Corollary 5.2}\\
&\quad\text{iff } \mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash Q]\!],
\end{aligned}$$

where the last equivalence follows from computational induction because $\mathrm{id}_A = \mathsf{fix}_{A \Rightarrow A}(\Phi) = \bigvee \langle \Phi^n(\lambda p : A.\, \bot)\rangle_n = \bigvee \langle \mathrm{Bot}^n \rangle_n$. ■

6. FULL ABSTRACTION

In this section we will prove the following full abstraction result.

THEOREM 6.1 (Full abstraction for the domain-theoretic interpretation). *In the domain-theoretic interpretation, $P \sim Q$ iff $\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash Q]\!]$ whenever $P, Q \in \mathcal{P}r_V$.*

Since the ω-birule $P = Q \iff (\forall n. P_{V,n} = Q_{V,n})$ whenever $P, Q \in \mathcal{P}r_V$ (where $P_{V,n}$ is the process $\mathrm{Nil}^n_V(\mathrm{Exp}^n_V(P)) \in \mathcal{NF}^n_V$) is admissible for strong late bisimulation (see Section 3) and for the equivalence induced by the close domain-theoretic interpretation (see Section 5.1), it suffices to prove

THEOREM 6.2 (Finite full abstraction for the domain-theoretic interpretation). *In the domain-theoretic interpretation, $P \sim Q$ iff $\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash Q]\!]$ whenever $P, Q \in \mathcal{P}r_V$ are finite.*

To interpret finite agents there is no need for recursion (in the metalanguage), so one can define also a set-theoretic interpretation, which uses the free-semilattice monad instead of Abramsky's powerdomain monad. The key technical result of this section is the existence of an *injective homomorphism* from the set-theoretic to the domain-theoretic interpretation. From this we have

THEOREM 6.3 (Equivalence on finite processes). *$\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash Q]\!]$ holds in the set-theoretic interpretation iff it holds in the domain-theoretic interpretation, whenever $P, Q \in \mathcal{P}r_V$ are finite.*

Therefore Theorem 6.1 follows from a much simpler full-abstraction result, namely

THEOREM 6.4 (Finite full abstraction for the set-theoretic interpretation). *In the set-theoretic interpretation, $P \sim Q$ iff $\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash Q]\!]$ whenever $P, Q \in \mathcal{P}r_V$ are finite.*

For the set-theoretic interpretation one can establish a stronger result:

THEOREM 6.5 (Universality of the set-theoretic interpretation). *If $A_0 \in \mathbf{Set}^{\mathcal{I}}$ is the object of agents in the set-theoretic interpretation, then $\mathcal{C}[\![V \vdash \_\,]\!]$ establishes a bijective correspondence $\mathcal{CNF}_V \cong A_0(\,|V|\,)$.*

The rest of this section is organized as follows: we define canonical interpretations of finite processes and establish some basic facts; then we prove Theorem 6.3; finally we prove Theorem 6.4 and 6.5. In Appendix B we recall some definitions regarding functors and monads used in the rest of this section.

6.1. Canonical interpretations of finite processes

In Section 4 we interpret processes in the category $\mathbf{Cpo}^{\mathcal{I}}$ (via translation into a metalanguage) using the object $A \overset{\text{def}}{=} \mu X.\, P(HX)$, where P is Abramsky's powerdomain monad and H is a suitable endofunctor on $\mathbf{Cpo}^{\mathcal{I}}$. To interpret finite processes there is no need of recursion (in the metalanguage), thus we can replace P with the free-semilattice monad M (in the interpretation of the metalanguage), and use the object $A_0 \overset{\text{def}}{=} \mu X.\, M(HX)$. A key difference between A and A_0 is that: A has a least element ($P(X)$ always has one), while A_0 is a *flat object*, i.e. the cpo $A_0(n)$ is flat for every $n \in \mathcal{I}$.

What follows identifies sufficient properties on a category $\mathcal{C}$ for defining *canonical* interpretations (of finite processes) in $\mathcal{C}^{\mathcal{I}}$, and gives ways for relating them. In particular, the interpretations of finite processes in the objects A and A_0 of $\mathbf{Cpo}^{\mathcal{I}}$ are canonical, and one can define a *canonical* interpretation of finite processes in $\mathbf{Set}^{\mathcal{I}}$ by analogy with that in A_0. We prove that there is an *homomorphism* $h : A_0 \to A$ from the set-theoretic interpretation in A_0 to the domain-theoretic interpretation in A, therefore two finite processes are identified in A, if they are identified in A_0.

DEFINITION 6.1. (OK category) We say that a category $\mathcal{C}$ is *OK* iff it is a biCCC with small limits and free semi-lattices. We say that a functor $\Delta : \mathcal{C}_1 \to \mathcal{C}_2$ (between OK categories) is *OK* iff it is full and faithful, it has a right adjoint U and a left adjoint c preserving finite products.

PROPOSITION 6.1. **Set** *and* **Cpo** *are OK. The functor $\Delta : \mathbf{Set} \to \mathbf{Cpo}$, mapping X to the flat cpo $(X, =_X)$, is OK.*

Proof. **Set** is obviously OK. **Cpo** is OK, in particular for the existence of free-semilattices see [3]. The functor Δ is clearly full and faithful. Δ has a right adjoint $U : \mathbf{Cpo} \to \mathbf{Set}$, mapping a cpo to its underlying set, and a left adjoint $c : \mathbf{Cpo} \to \mathbf{Set}$, mapping a cpo to the set of its connected components. It is easy to show that c preserves finite products. ∎

PROPOSITION 6.2. *If $\Delta : \mathcal{C}_1 \to \mathcal{C}_2$ is OK, then it preserves limits, colimits, exponentials, and free-semilattices. Moreover, $\mathcal{C}_1$ is an exponential ideal of $\mathcal{C}_2$.*

Proof. Limits and colimits are preserved because Δ has left and right adjoints. $\mathcal{C}_1$ (as a full sub-category of $\mathcal{C}_2$) is an exponential ideal, i.e. $Y^X \in \mathcal{C}_1$ whenever $Y \in \mathcal{C}_1$ and $X \in \mathcal{C}_2$. More precisely, $(\Delta Y)^X$ is canonically isomorphic to $\Delta(Y^{c(X)})$, because the left adjoint c preserves finite products. Exponentials are preserved, because $\mathcal{C}_1$ is an exponential ideal and $c(\Delta X)$ is canonically isomorphic to X (as Δ is full and faithful). Let $\mathbf{SL}(\mathcal{C}_i)$ be the category of semilattices in $\mathcal{C}_i$, $F_i : \mathbf{SL}(\mathcal{C}_i) \to \mathcal{C}_i$ the forgetful functor, and $M_i \dashv F_i$ the adjunction establishing that $\mathcal{C}_i$ has free-semilattices. The adjunction $\Delta \dashv U$ lifts to semi-lattices, because Δ and U preserve finite products. Now consider the following adjunctions

$$\begin{tikzcd}
\mathbf{SL}(\mathcal{C}_1) \arrow[r, shift left, "F_1"'] \arrow[r, phantom, "\perp"] \arrow[d, shift right, "\Delta"'] \arrow[d, phantom, "\dashv"] & \mathcal{C}_1 \arrow[l, shift left, "M_1"'] \arrow[d, shift left, "\Delta"] \arrow[d, phantom, "\vdash"] \\
\mathbf{SL}(\mathcal{C}_2) \arrow[r, shift left, "F_2"] \arrow[r, phantom, "\top"] \arrow[u, shift right, "U"'] & \mathcal{C}_2 \arrow[l, shift left, "M_2"] \arrow[u, shift left, "U"]
\end{tikzcd}$$

Since the inner square commutes (up to isomorphism), so does the outer square. This means that $\Delta(M_1 X) = M_2(\Delta X)$, i.e. Δ preserves free-semilattices. ∎

PROPOSITION 6.3. *For any small category $\mathcal{W}$*

- *the category $\mathcal{C}^{\mathcal{W}}$ is OK, if $\mathcal{C}$ is;*
- *the functor $\Delta^{\mathcal{W}}$ is OK, if $\Delta : \mathcal{C}_1 \to \mathcal{C}_2$ is.*

Proof. The structure which makes $\mathcal{C}^{\mathcal{W}}$ OK is defined pointwise, except for exponentials, which are defined in terms of small limits and exponentials in $\mathcal{C}$. The left and right adjoints of $\Delta^{\mathcal{W}}$ are $U^{\mathcal{W}}$ and $c^{\mathcal{W}}$, and they inherit the necessary properties from Δ, U and c. ∎

The notion of algebra for an endofunctor or a monad is well-known (see Appendix B). We consider the more general case of objects with two (or more) algebra structures.

DEFINITION 6.2. Given F and G monads or endofunctors on a category $\mathcal{C}$, we say that $a = (a^F, a^G)$ is an *(F, G)-algebra* iff $a^F : FA \to A$ is an F-algebra and $a^G : GA \to A$ is a G-algebra. Moreover, we say that $h : A \to B$ is an (F,G)-*homomorphism* from a to b iff h is both an F-homomorphism and G-homomorphism. We write $\mathcal{C}^{(F,G)}$ for the category of (F,G)-algebras.

PROPOSITION 6.4.

1.*If M is a monad and H an endofunctor on $\mathcal{C}$, then every initial MH-algebra $A \overset{\text{def}}{=} \mu X.M(HX)$ (if it exists) is an initial (M,H)-algebra as follows*

$$\begin{tikzcd} HA \arrow[r, "\eta^M_{HA}"] & M(HA) \cong A & M^2(HA) \cong MA \arrow[l, "\mu^M_{HA}"'] \end{tikzcd}$$

2.*If T is a monad and $\lambda : M \dot{\to} T$ a monad morphism, then any $X \cong T(HX)$ is an (M,H)-algebra as follows*

$$\begin{tikzcd} HX \arrow[r, "\eta^T_{HX}"] & T(HX) \cong X & T^2(HX) \cong TX \arrow[l, "\mu^T_{HX}"'] & MX \arrow[l, "\lambda_X"'] \end{tikzcd}$$

Proof. (1) Let $b = (b^M : MB \to B, b^H : HB \to B)$ be an (M,H)-algebra. B is also an MH-algebra with $b^{MH} \overset{\text{def}}{=} b^M \circ M(b^H) : M(HB) \to B$. The unique MH-homomorphism $h : A \to B$ is also an (M,H)-homomorphism, because

$$\begin{tikzcd} HA \arrow[r, "\eta^M_{HA}"] \arrow[d, "Hh"'] & M(HA) \arrow[r, "\sim"] \arrow[d, "M(Hh)"'] & A \arrow[d, "h"'] & M(HA) \arrow[l, "\sim"'] \arrow[d, "M(Hh)"'] & M^2(HA) \arrow[l, "\mu^M_{HA}"'] \arrow[r, "\sim"] \arrow[d, "M(Hh)"'] & MA \arrow[d, "Mh"'] \\ HB \arrow[r, "\eta^M_{HB}"'] & M(HB) \arrow[r, "b^{MH}"'] & B & M(HB) \arrow[l, "b^{MH}"] & M^2(HB) \arrow[l, "\mu^M_{HB}"] \arrow[r, "Mb^{MH}"'] & MB \end{tikzcd}$$

commute and the following equations hold (as b^M is an M-algebra)

- $b^{MH} \circ \eta^M_{HB} = b^M \circ M(b^H) \circ \eta^M_{HB} = b^M \circ \eta^B \circ b^H = b^H$,
- $b^{MH} \circ \mu^M_{HB} = b^M \circ M(b^H) \circ \mu^M_{HB} = b^M \circ \mu^B \circ M^2(b^H) = b^M \circ M(b^M) \circ M^2(b^H) = b^M \circ M(b^{MH})$.

Moreover h is the unique (M,H)-homomorphism, because any (M,H)-homomorphism $k : A \to B$ is also an MH-homomorphism (and therefore equal to h)

$$\begin{tikzcd} M(HA) \arrow[r, "M\eta^M_{HA}"] \arrow[d, "M(Hk)"'] & M^2(HA) \cong MA \arrow[r, "\mu^M_{HA}"] \arrow[d, "Mk"'] & M(HA) \cong A \arrow[d, "k"'] \\ M(HB) \arrow[r, "Mb^H"] & MB \arrow[r, "b^M"] & B \end{tikzcd}$$

■

Theorem 6.6. *If $\mathcal{C}$ is OK, then any (M,H)-algebra a on A in $\mathcal{C}^{\mathcal{I}}$ induces a* canonical *interpretation $\mathcal{C}[\![V \vdash P]\!]_a \in A(\,|V|\,)$ for any finite $P \in \mathcal{P}r_V$, where*

- *M is the free-semilattice monad on $\mathcal{C}^{\mathcal{I}}$;*
- *H is the endofunctor on $\mathcal{C}^{\mathcal{I}}$ s.t. $HX = N \times (N \Rightarrow X) + N \times N \times X + N \times \delta X + X$;*
- *N is the object of $\mathcal{C}^{\mathcal{I}}$ s.t. $N(n) = n$, i.e. the coproduct of n copies of 1;*
- *δ is the endofunctor on $\mathcal{C}^{\mathcal{I}}$ s.t. $(\delta X)(n) = X(n+1)$.*

Proof. We have to interpret the term constructors for finite processes (see Section 4.6).

- Since a^M is an M-algebra, A has a semilattice structure, which provides the interpretation of $\mathsf{nil} : A$ and $\mathsf{sum} : A, A \to A$;
- N has a decidable equality, so one can interpret (using also the semilattice structure on A) $\mathsf{M} : N, N, A \to A$ and $\mathsf{MM} : N, N, A \to A$;
- Finally using the H-algebra structure a^H (and the morphism $\lambda f : N \Rightarrow X.\,\delta f\,\mathsf{new} : (N \Rightarrow X) \to \delta X$ for $X = A$) one can interpret $\mathsf{in} : N, (N \Rightarrow A) \to A$, $\mathsf{out} : N, N, A \to A$, $\mathsf{bout} : N, (N \Rightarrow A) \to A$ and $\mathsf{tau} : A \to A$.

■

Theorem 6.7. *If $\mathcal{C}$ is OK, M and H are the endofunctors on $\mathcal{C}^{\mathcal{I}}$ considered in Theorem 6.6 and $h : a_1 \to a_2$ is an (M,H)-homomorphism, then*

$$h_{|V|}(\mathcal{C}[\![V \vdash P]\!]_{a_1}) = \mathcal{C}[\![V \vdash P]\!]_{a_2}$$

for any finite $P \in \mathcal{P}r_V$.

Proof. It is immediate from the interpretation of term constructors that h commutes with them, i.e. $h \circ op_1 = op_2 \circ O(h)$ where $op_i : O(A_i) \to A_i$ is the interpretation in A_i of the term constructor $op : O(A) \to A$. This property is inherited by the interpretation of derived term constructors, in particular by the (open) interpretation of finite processes. ■

Theorem 6.8. *If $\Delta : \mathcal{C}_1 \to \mathcal{C}_2$ is OK, M_i and H_i are the endofunctors on $\mathcal{C}_i^{\mathcal{I}}$ considered in Theorem 6.6, and a is an (M_1,H_1)-algebra on A, then*

- *a is also an (M_2,H_2)-algebra in $\mathcal{C}_2^{\mathcal{I}}$, by considering $\mathcal{C}_1^{\mathcal{I}}$ as a full sub-category of $\mathcal{C}_2^{\mathcal{I}}$ via the embedding $\Delta^{\mathcal{I}}$;*
- *$\mathcal{C}[\![V \vdash P]\!]_1 = \mathcal{C}[\![V \vdash P]\!]_2 \in A(\,|V|\,)$ for any finite $P \in \mathcal{P}r_V$, where $\mathcal{C}[\![V \vdash P]\!]_i$ is the canonical interpretation in $\mathcal{C}_i^{\mathcal{I}}$ induced by the (M_i,H_i)-algebra a.*

Proof. Since $\Delta^{\mathcal{I}}$ is OK, it preserves the categorical structure for defining canonical interpretations. In particular, $\Delta^{\mathcal{I}}$ commutes with M and H, and so it preserves (M,H)-algebras. ■

The following corollaries relate the set-theoretic and domain-theoretic interpretation.

LEMMA 6.1. *If $\Delta : \mathcal{C}_1 \to \mathcal{C}_2$ has a right adjoint U and the diagram*

$$\begin{tikzcd}
\mathcal{C}_1 \arrow[r, "\Delta"] \arrow[d, "F_1"'] & \mathcal{C}_2 \arrow[d, "F_2"] \\
\mathcal{C}_1 \arrow[r, "\Delta"'] & \mathcal{C}_2
\end{tikzcd}$$

commutes, then Δ maps initial F_1-algebras to initial F_2-algebras.

Proof. Given an initial F_1-algebra $a : F_1 A \to A$, we sketch the proof of initiality for the F_2-algebra $\Delta a : F_2(\Delta A) = \Delta(F_1 A) \to \Delta A$. Namely, for any $b : F_2 B \to B$ we define the unique F_2-homomorphism $f : \Delta A \to B$ using the adjunction $\Delta \dashv U$:

- the F_2-algebra b induces an F_1-algebra $b' : F_1(UB) \to UB$ defined as the composite

$$F_1(UB) \xrightarrow{\eta_{F_1(UB)}} U(\Delta(F_1(UB))) = U(F_2(\Delta(UB))) \xrightarrow{U(F_2\epsilon_B)} U(F_2 B) \xrightarrow{Ub} UB$$

  where η and ϵ are the unit and counit of the adjunction $\Delta \dashv U$;
- by initiality of a one has an F_1-homomorphism $f' : A \to UB$ from a to b';
- by the natural isomorphism $\mathcal{C}_1(A, UB) \stackrel{\cdot}{\cong} \mathcal{C}_2(\Delta A, B)$ one gets $f : \Delta A \to B$.

One finally checks that f has the required properties. ■

COROLLARY 6.1. *The canonical interpretations in the initial MH-algebras in* $\mathbf{Set}^{\mathcal{I}}$ *and* $\mathbf{Cpo}^{\mathcal{I}}$ *coincide.*

Proof. By Theorem 6.8 the (M, H)-algebra on the initial MH-algebra $A_0 \stackrel{\text{def}}{=} \mu X. M(HX)$ in $\mathbf{Set}^{\mathcal{I}}$ is also an (M, H)-algebra in $\mathbf{Cpo}^{\mathcal{I}}$, that induces the same interpretation of finite processes. But $\Delta^{\mathcal{I}}$ commutes with M and H, therefore by Lemma 6.1 $\Delta^{\mathcal{I}}$ maps A_0 to the initial MH-algebra in $\mathbf{Cpo}^{\mathcal{I}}$. ■

COROLLARY 6.2. *The domain-theoretic interpretation in $A \stackrel{\text{def}}{=} \mu X. P(HX)$ of* $\mathbf{Cpo}^{\mathcal{I}}$ *(see Section 4.6) is canonical. Furthermore, it is related to the canonical interpretation in $A_0 \stackrel{\text{def}}{=} \mu X. M(HX)$ by an (M, H)-homomorphism $h : A_0 \to A$.*

Proof. The domain-theoretic interpretation of finite processes in A is canonical because it coincides with the canonical interpretation induced by the (M, H)-algebra on A given by Proposition 6.4 by taking $T = P$ and λ_X as the (unique) semilattice homomorphism $MX \to PX$ induced by $\eta^P_X : X \to PX$. This is proved by showing that the interpretation of the term constructors for finite processes (see Section 4.6) coincide. Existence (and uniqueness) of $h : A_0 \to A$ follows immediately from Proposition 6.4. ■

6.2. Injectivity and equivalence

To prove that the canonical interpretations in A_0 and A induce the same equivalence on finite processes, we show that the (M, H)-homomorphism $h : A_0 \to A$ of Corollary 6.2 is injective. In what follows $\mathcal{C}$ is OK, and L is a monad on $\mathcal{C}$ s.t. the unit η^L is monic and the following equations (expressed in the metalanguage) hold

- $\mathsf{let}_L\, x_1, x_2 \Leftarrow c_1, c_2\, \mathsf{in}\, f(x_1, x_2) = \mathsf{let}_L\, x_2, x_1 \Leftarrow c_2, c_1\, \mathsf{in}\, f(x_1, x_2)$,
- $\mathsf{let}_L\, x_1, x_2 \Leftarrow c, c\, \mathsf{in}\, f(x_1, x_2) = \mathsf{let}_L\, x \Leftarrow c\, \mathsf{in}\, f(x, x)$.

Moreover, we assume that the free-semilattice monad M and the monad P exist and are s.t. the category $\mathcal{C}^P$ of P-algebras is isomorphic to the category $\mathcal{C}^{(L,M)}$ of (L, M)-algebras.

Remark. The intended interpretation for L is the lifting monad on **Cpo**. In this case P is Abramsky's powerdomain. The equations for L say that: changing the order of computation and recomputing are irrelevant. By definition of P, every P-algebra structure induces an M-algebra structure on the same object of $\mathcal{C}$, and P-homomorphisms are also M-homomorphisms. This amounts to have a monad morphism $\lambda : M \dot{\to} P$.

Notation. If T is a monad, we write η^T and μ^T for its unit and multiplication, moreover when X and Y have a given T-algebra structure (which is clear from the context) we write $f : X \to_T Y$ to mean that f is a T-homomorphism.

DEFINITION 6.3. Given an operation $op : X^n \to X$, its *extension* $op' : (LX)^n \to LX$ to LX is given by $op'(\bar{c}) \triangleq \mathsf{let}\, \bar{x} \Leftarrow \bar{c}\, \mathsf{in}\, [op(\bar{x})]$.

LEMMA 6.2. *An M-algebra structure on X induces a P-algebra structure on LX. Moreover, if $f : X \to_M Y$ then $Lf : LX \to_P LY$, and $\eta^L_X : X \to_P LX$ and $\mu^L_X : L^2X \to_P LX$.*

Proof. An M-algebra structure on X means that we have a semilattice $(X, 0, \cup)$. LX has a free L-algebra structure (namely μ^L_X), therefore to define a P-algebra structure on LX it suffices to have a semilattice. By the definition of extension and the equational properties of L, it is immediate to see that $(LX, 0', \cup')$ is a semilattice and the three morphisms are homomorphisms. ■

THEOREM 6.9. *Let H be an endofunctor on $\mathcal{C}$ and $\sigma : HL \dot{\to} LH$ a distributive law. If the initial algebras $A \stackrel{\text{def}}{=} \mu X.P(HX)$ and $A_0 \stackrel{\text{def}}{=} \mu X.M(HX)$ exist, then the unique (M, H)-homomorphism $h : A_0 \to A$ of Proposition 6.4 is monic.*

Proof. We define a map $p : A \to LA_0$ and prove that $p \circ h = \eta^L_{A_0}$. Then h is monic, because η^L_X is monic (by assumption). In what follows

- $\lambda : M \dot{\to} P$ is the monad morphism from M to P;
- $a^H_0 : HA_0 \to A_0$ is the H-algebra structure on A_0;

- $a_0^M : MA_0 \to A_0$ is the (free) M-algebra structure on A_0;
- $a_0^P : P(LA_0) \to LA_0$ is the P-algebra structure on LA_0 induced by a_0^M as given in Lemma 6.2.

Recall that $h : A_0 \to A$ is the unique MH-homomorphism s.t. the diagram

$$\begin{tikzcd}
M(HA_0) \arrow[rr, "\sim"] \arrow[d, "M(Hh)"'] & & A_0 \arrow[d, "h"] \\
M(HA) \arrow[r, "\lambda_{HA}"] & P(HA) \arrow[r, "\sim"] & A
\end{tikzcd}$$

commutes, and let $p : A \to LA_0$ be the unique PH-homomorphism s.t. the diagram

$$\begin{tikzcd}
P(HA) \arrow[rrr, "\sim"] \arrow[d, "P(Hp)"'] & & & A \arrow[d, "p"] \\
P(H(LA_0)) \arrow[r, "P\sigma_{A_0}"] & P(L(HA_0)) \arrow[r, "P(La_0^H)"] & P(LA_0) \arrow[r, "a_0^P"] & LA_0
\end{tikzcd}$$

commutes.

The composite $p \circ h$ is the unique MH-homomorphism $e : A_0 \to LA_0$ making the following diagram commute:

$$\begin{tikzcd}
M(HA_0) \arrow[rrrr, "\sim"] \arrow[d, "M(He)"'] & & & & A_0 \arrow[d, "e"] \\
M(H(LA_0)) \arrow[r, "\lambda_{H(LA_0)}"] & P(H(LA_0)) \arrow[r, "P\sigma_{A_0}"] & P(L(HA_0)) \arrow[r, "P(La_0^H)"] & P(LA_0) \arrow[r, "a_0^P"] & LA_0
\end{tikzcd}$$

and we prove that $p \circ h = \eta^L_{A_0}$ by establishing the commutativity of:

$$\begin{tikzcd}
M(HA_0) \arrow[rrrr, "\sim"] \arrow[d, "M(H\eta^L_{A_0})"'] & & & & A_0 \arrow[d, "\eta^L_{A_0}"] \\
M(H(LA_0)) \arrow[r, "\lambda_{H(LA_0)}"] & P(H(LA_0)) \arrow[r, "P\sigma_{A_0}"] & P(L(HA_0)) \arrow[r, "P(La_0^H)"] & P(LA_0) \arrow[r, "a_0^P"] & LA_0
\end{tikzcd}$$

which follows from the commutativity of the diagram below

$$\begin{tikzcd}
M(HA_0) \arrow[r, equal] \arrow[d, "M(H\eta^L_{A_0})"'] \arrow[dr, phantom, "(1)"] & M(HA_0) \arrow[r, "Ma^H_0"] \arrow[d, "M(\eta^L_{HA_0})"] \arrow[dr, phantom, "(2)"] & MA_0 \arrow[r, "a^M_0"] \arrow[d, "M\eta^L_{A_0}"'] \arrow[dr, phantom, "(3)"] & A_0 \arrow[dd, "\eta^L_{A_0}"] \\
M(H(LA_0)) \arrow[r, "M\sigma_{A_0}"] \arrow[d, "\lambda_{H(LA_0)}"'] \arrow[dr, phantom, "(4)"] & M(L(HA_0)) \arrow[r, "M(La^H_0)"] \arrow[d, "\lambda_{L(HA_0)}"] \arrow[dr, phantom, "(4')"] & M(LA_0) \arrow[d, "\lambda_{LA_0}"] & \\
P(H(LA_0)) \arrow[r, "P\sigma_{A_0}"'] & P(L(HA_0)) \arrow[r, "P(La^H_0)"'] & P(LA_0) \arrow[r, "a^P_0"'] & LA_0
\end{tikzcd}$$

where (1) commutes by a property of σ, viz. $\sigma_X \circ H\eta^L_X = \eta^L_{HX}$; (2) commutes by naturality of η^L; (3) commutes because $\eta^L_X : X \to_M LX$ (see Lemma 6.2); and (4) and (4′) commute by naturality of λ. ■

To finally prove Theorem 6.3, we want to apply Theorem 6.9 when H is the endofunctor on $\mathcal{C}^{\mathcal{I}}$ considered in Theorem 6.6 and the monads are the pointwise extensions to $\mathcal{C}^{\mathcal{I}}$ of the monads M, P and L on $\mathcal{C}$, whose existence is assumed at the beginning of Section 6.2. (Note that these pointwise extensions inherit the properties stated at the beginning of Section 6.2 for the original monads.) Thus, we need provide a distributive law

$$HL^{\mathcal{I}} \dot{\to} L^{\mathcal{I}}H$$

where $HX = N\times(N\Rightarrow X) + N\times N\times X + N\times\delta X + X$ and $(L^{\mathcal{I}}X)(n) = (LX)(n)$. This can be done using the following distributive laws:

- $\kappa_{\eta^{L^{\mathcal{I}}}_N} : K_N L^{\mathcal{I}} \dot{\to} L^{\mathcal{I}} K_N$, where $K_N(X) = N$ and $(\kappa_{\eta^{L^{\mathcal{I}}}_N})_X = \eta^{L^{\mathcal{I}}}_N$.
- id : $\delta L^{\mathcal{I}} \dot{\to} L^{\mathcal{I}}\delta$, noting that $(\delta(L^{\mathcal{I}}X))(n) = L(X(n+1)) = (L^{\mathcal{I}}(\delta X))(n)$.
- $\sigma_X : (N \Rightarrow L^{\mathcal{I}}X) \dot{\to} L^{\mathcal{I}}(N \Rightarrow X)$ defined, using Proposition 4.2, at stage n as the mapping

$$\begin{array}{rcl} (L(Xn))^n \times L(X(n+1)) & \to & L((Xn)^n \times X(n+1)) \\ \bar{c} & \mapsto & \mathsf{let}\ \bar{x} \Leftarrow \bar{c}\ \mathsf{in}\ [\bar{x}] \end{array}$$

- id : $\mathrm{Id}\, L^{\mathcal{I}} \dot{\to} L^{\mathcal{I}}\mathrm{Id}$.

together with the constructions on distributive laws given below:

- If $\sigma_1 : H_1 L^{\mathcal{I}} \dot{\to} L^{\mathcal{I}} H_1$ and $\sigma_2 : H_2 L^{\mathcal{I}} \dot{\to} L^{\mathcal{I}} H_2$ are distributive laws then so is the composite

$$(H_1 \times H_2)L^{\mathcal{I}} = H_1 L^{\mathcal{I}} \times H_2 L^{\mathcal{I}} \xrightarrow{\sigma_1 \times \sigma_2} L^{\mathcal{I}} H_1 \times L^{\mathcal{I}} H_2 \xrightarrow{\Psi} L^{\mathcal{I}}(H_1 \times H_2)$$

where $\Psi(c_1, c_2) = \mathsf{let}\ \langle x_1, x_2\rangle \Leftarrow \langle c_1, c_2\rangle\ \mathsf{in}\ [\langle x_1, x_2\rangle]$.

• If $\sigma_1 : H_1 L^{\mathcal{I}} \dot{\to} L^{\mathcal{I}} H_1$ and $\sigma_2 : H_2 L^{\mathcal{I}} \dot{\to} L^{\mathcal{I}} H_2$ are distributive laws then so is the composite

$$(H_1 + H_2)L^{\mathcal{I}} = H_1 L^{\mathcal{I}} + H_2 L^{\mathcal{I}} \xrightarrow{\sigma_1 + \sigma_2} L^{\mathcal{I}} H_1 + L^{\mathcal{I}} H_2 \xrightarrow{[L^{\mathcal{I}}\amalg_1, L^{\mathcal{I}}\amalg_2]} L^{\mathcal{I}}(H_1 + H_2)$$

6.3. Finite full abstraction and universality

By Proposition 3.4: $P \sim Q$ iff $\mathrm{CNF}_V(P) \equiv \mathrm{CNF}_V(Q)$ whenever $P, Q \in \mathcal{P}r_V$ are finite. Moreover, in the closed set-theoretic interpretation we have that $\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash \mathrm{CNF}_V(P)]\!]$ whenever $P \in \mathcal{P}r_V$ is finite. In fact:

- The axioms $\mathcal{A}_\pi$ are valid in the close domain-theoretic interpretation (see Section 5).
- $\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash \mathrm{CNF}_V(P)]\!]$ in the close domain-theoretic interpretation whenever $P \in \mathcal{P}r_V$ is finite, because $P = \mathrm{CNF}_V(P)$ is derivable from $\mathcal{A}_\pi$ (see Lemma 3.2).
- $\mathcal{C}[\![V \vdash P]\!] = \mathcal{C}[\![V \vdash \mathrm{CNF}_V(P)]\!]$ in the close set-theoretic interpretation whenever $P \in \mathcal{P}r_V$ is finite, because of Theorem 6.3.

By the observations above Theorem 6.4 follows from Theorem 6.5. Finally, the bijective correspondence between $\mathcal{CNF}_V$ and $A_0(\,|V|\,)$ is proved by observing that

$$\mathcal{CNF}_V = \bigcup_{n\in\omega} \mathcal{CNF}_V^n \qquad A_0(\,|V|\,) = \bigcup_{n\in\omega} A_0^n(\,|V|\,)$$

and the inductive definitions of $\mathcal{CNF}_V^n$ and $A_0^n(\,|V|\,)$ are the *same*:

- the set $\mathcal{CNF}_V^n$ of canonical normal forms of depth less than n and names in V is defined by induction on n as follows $\mathcal{CNF}_V^0 \overset{\mathrm{def}}{=} \emptyset$, $\mathcal{CNF}_V^{n+1} \overset{\mathrm{def}}{=} \mathcal{P}_{fin}(p\mathcal{CNF}_V^n)$, and the set $p\mathcal{CNF}_V^n$ of *prefixed* canonical normal forms is given by the grammar

$$\begin{array}{rcl} p\mathcal{CNF}_V^n & ::= & a(c).\left(\sum_{a\in V}[c=a]\mathcal{CNF}_V^n + [c \notin V]\mathcal{CNF}_{V\cup\{c\}}^n\right) \;\Big|\; \\ & & \overline{a}b.\mathcal{CNF}_V^n \;\Big|\; \overline{a}(c).\mathcal{CNF}_{V\cup\{c\}}^n \;\Big|\; \tau.\mathcal{CNF}_V^n \end{array}$$

  where $n \in \omega$, $a, b \in V$ and c is the *first* name not in V.
- the set $A_0^n(\,|V|\,)$ is the object $(MH)^n(0)$ of $\mathbf{Set}^{\mathcal{I}}$ at stage $|\,V\,|$, or equivalently it can be defined by induction on n as follows: $A_0^0(v) \overset{\mathrm{def}}{=} \emptyset$ and

$$A_0^{n+1}(v) \overset{\mathrm{def}}{=} \mathcal{P}_{fin}(v \times (A_0^n(v))^v \times A_0^n(v+1) + v \times v \times A_0^n(v) + v \times A_0^n(v+1) + A_0^n(v))$$

  where $n, v \in \omega$.

7. BISIMILARITY UNDER A CONSTRAINT

Distinctions, finite symmetric and irreflexive binary relation on names [21], allow us to forbid certain identifications of names. They are useful in the π-calculus because in it ports, variables and constants are not distinguished —they are all just *names.* Distinctions express permanent inequalities on names; thus (a,b) being in a distinction means that a must be kept separate from b. If D is a distinction then $P \sim^D Q$ means that P and Q are bisimilar under all substitutions which respect D.

In this section we show that the theory of the previous sections can be easily generalised to handle distinctions. Actually, we shall be more general and consider *any decidable property* on a finite list V of distinct names, which we call *constraints on* V or, simply, *constraints* when V is clear from the context. Formally, a constraint on V, say ϕ, is a subset of $\mathcal{N}^V$ (the functions from the *underlying set* of V to $\mathcal{N}$) closed under injective substitutions (i.e. such that for every injective substitution ι, if $s \in \phi$ then $\iota s \in \phi$). We use ϕ to range over constraints, and call $\mathcal{N}^V$ the *maximal* constraint. If ϕ is a constraint on V and σ a substitution, then σ *respects* ϕ if $\sigma \upharpoonright V \in \phi$.

Definition 7.1. (Bisimulation under a constraint) Let ϕ be a constraint on V. For $P, Q \in \mathcal{P}r_V$, we set $P \sim^\phi Q$ (read "P bisimilar to Q under the constraint ϕ") if $P\sigma \sim Q\sigma$ for all substitutions σ that respect ϕ.

Constraints are more expressive than distinctions, in that they allow us to express more refined forms of process bisimilarities. For instance, using some obvious logical connectives for a concise description of constraints, $P \sim^{[a\neq b]\vee[a\neq c]} Q$ says that P and Q are equivalent under all substitutions which keep a different from b or c, and $P \sim^{[a=b]\vee[a=c]} Q$ says that P and Q are equivalent under all substitutions which make a equal to b or c. Late congruence is, by definition, the same as bisimilarity under the maximal constraint, whereas late bisimilarity can be proved to coincide with bisimilarity under the constraint made of all injective substitutions. All processes are bisimilar under the empty constraint.

The set of constraints (on a fixed list of names) ordered by inclusion is a complete boolean algebra. Every pair of processes $P, Q \in \mathcal{P}r_V$ has an *optimal* constraint $\Theta_{(P,Q)} \stackrel{\text{def}}{=} \bigcup\{\phi \mid P \sim^\phi Q\}$ such that $P \sim^\phi Q$ iff $\phi \subseteq \Theta_{(P,Q)}$. In other words, the optimal constraint collects the necessary conditions on names for the equality between two processes to hold. It may be that $\Theta_{(P,Q)}$ is the false constraint (e.g. for $P \stackrel{\text{def}}{=} \tau.\mathbf{0}$ and $Q \stackrel{\text{def}}{=} \overline{a}b.\mathbf{0}$) in which case no choice of names can make P and Q bisimilar.

The notion of constraint was suggested by the model as for any V we have a bijective correspondence between "constraints on V" and "natural transformations $N^{|V|} \to 2$". For $V \equiv a_0, \ldots, a_{n-1}$, a constraint ϕ and a natural transformation f are in the bijective correspondence whenever $f_m(i_0, \ldots, i_{n-1}) = \text{tt}$ $(0 \leq i_j < m)$ iff $[a_j \mapsto \underline{i_j}]_{0\leq j<n} \in \phi$ where we write $\underline{i}$ for the i^{th} name in $\mathcal{N}$. It is then immediate to define an *interpretation under a constraint*. By pullback, we *objectify* constraints ϕ as $\Phi \stackrel{\text{def}}{=} \chi(\phi)^{-1}(\mathbf{tt})$ where $\chi(\phi)$ is the natural transformation corresponding to ϕ; and, by restriction, we define

$$\mathcal{O}_\phi[\![V \vdash \_]\!] \stackrel{\text{def}}{=} (\Phi \subseteq N^{|V|} \xrightarrow{\mathcal{O}[\![V\vdash\_]\!]} A). \tag{32}$$

When ϕ is the maximal constraint, (32) specialises to the open interpretation, whilst when ϕ is the constraint consisting of all injective substitutions (32) is essentially the closed interpretation.

Full abstraction for bisimilarity under constraint follows as a corollary of full abstraction for late bisimilarity (Theorem 6.1).

COROLLARY 7.1 (Full abstraction of the domain-theoretic interpretation under constraints). *Let ϕ be a constraint on V. In the domain-theoretic interpretation, for $P, Q \in \mathcal{P}r_V$, $P \sim^{\phi} Q$ iff $\mathcal{O}_{\phi}[\![V \vdash P]\!] = \mathcal{O}_{\phi}[\![V \vdash Q]\!]$.*

Similarly to what has been done for the closed semantics (Lemma 4.1), the interpretation under constraints can be proved to be compositional. The congruence properties of the associated notion of bisimilarity follow.

8. CONCLUSIONS

The denotational model is superior to the operational approach in revealing a few basic properties of $\sim$ and $\sim^{\mathrm{c}}$. A good example is the invariance of $\sim$ under injective substitutions, which is a straightforward consequence of (30) —the operational proof, although not difficult, is rather tedious. Other examples, are the congruence properties of $\sim$ and $\sim^{\mathrm{c}}$ (e.g. that $\sim^{\mathrm{c}}$ is preserved by all operators, and that $\sim$ is preserved by all operators but input), which follow directly from the definitions of $\mathcal{O}[\![\_]\!]$ and $\mathcal{C}[\![\_]\!]$.

The denotational model is also interesting for proving basic laws of the operators of π-calculus, like associativity of parallel composition and the extrusion law for restriction "$\boldsymbol{\nu} a\,(P \mid Q) \sim^{\mathrm{c}} P \mid \boldsymbol{\nu} a\, Q$ if $a \notin \mathrm{fn}(P)$". The operational proofs of these laws in [21] require some ingenuity (e.g. the "bisimilarity up to bisimilarity and up to restriction" technique). For an idea of how these proofs may be carried over in the denotational model, see the validation of laws in Subsection 5.1.

Guarded replication. In our π-calculus syntax the plain replication $!P$ has been replaced by a guarded replication $!\alpha.\, P$. This simplification is justified by the laws of π-calculus strong bisimilarity [29] and allows us to avoid the issue of divergence. For instance, with plain replication a denotational semantics would validate $!\mathbf{0} = \bot$, whilst bisimulation validates $!\mathbf{0} = \mathbf{0}$.

Matching/mismatching. For describing the canonical forms induced by our model and hence to obtain the universality Theorem 6.5 we need both matching and mismatching —operators whose theoretical and pragmatic relevance for π-calculus is often debated. The importance of these constructs for equational reasoning had already been expressed in [25].

The metalanguage. We have factored the denotational semantics for the π-calculus through a metalanguage suggested by model-theoretic considerations.

The metalanguage can easily cope with operators not in the π-calculus syntax. For instance, interrupt operators like those in LOTOS [8], which are *not definable* in the π-calculus —even up to weak bisimulation.

An interesting direction of research is to turn the metalanguage into a typed higher order process calculus, with an operational semantics and notion of bisimulation conservatively extending those of the π-calculus.

The translation of the π-calculus in the metalanguage uses a type of agents $A = P(HA)$, where P is Abramsky's powerdomain monad and H is an endofunctor corresponding to the action capabilities. It is conceivable to replace P with some other monad and H with some other endofunctor. This flexibility allows to accommodate smoothly other languages, e.g.

- to deal with global variables one should replace P with the monad $TX \overset{\text{def}}{=} P(X \times S)^S$, where S is an object of states;
- to deal with the πI-calculus [31] (a symmetric sub-calculus of π-calculus where the free-output construct is forbidden and hence only private names can be exchanged), one should use the endofunctor $HX \overset{\text{def}}{=} N \times \delta X + N \times \delta X + X$. This gives a fully abstract model for πI-bisimilarity; the proof mimics the one outlined in Section 6.

In particular, by changing H we can define a denotational semantics in $\mathbf{Cpo}^{\mathcal{I}}$ for languages ranging from pure CCS to Higher-Order π-calculus (HOπ) [28], along the lines outlined for the π-calculus.

Also the proof of full abstraction is fairly reusable. It copes with the polyadic π-calculus (where several names can be sent at once) and value-passing CCS (with binary sums and guarded recursion), provided the set of values is finite; but it cannot cope with HOπ and CCS with *infinite*-value passing.

We have obtained full abstraction with respect to a *simple* minded domain-theoretic model (in comparison to other categories proposed for Algol-like languages). The main problem to get full abstraction results for domain-theoretic models is not local names and mobility, but higher order! In fact, *simple* domain-theoretic models cannot achieve full abstraction for PCF nor for the $\lambda\nu$-calculus (a CBV λ-calculus whose base types are *bool* and *unit ref*, see [26]). Therefore, to get full abstraction results for higher-order calculi with static binding, like Plain CHOCS [35] and HOπ, one should consider more refined models (probably based on *game semantics* [2, 18]). Thomsen [35], Hennessy [14], and Jeffrey [19] have given denotational models for higher-order process calculi with *dynamic binding* on names. But their constructions cannot account for calculi based on static binding like the π-calculus, Plain CHOCS, and HOπ.

Bisimilarity under a constraint. We have seen that the set of bisimilarities under constraints has a rich structure —that of a complete boolean algebra— and that any pair of processes has an optimal constraint for bisimilarity. It might be interesting to explore this structure in more detail and see whether on non-trivial subsets of π-calculus the optimal constraint of two processes can be computed efficiently.

Limits of our approach. Our semantics for π-calculus is a special case of a uniform approach to give semantics to a variety of calculi. However, this approach deals only with *strong late bisimulation*, which from a denotational point of view appears to be the simplest equivalence to handle (amongst those proposed for the π-calculus). We do not know how to capture denotationally other equivalences:

- *Weak bisimulation* —where internal actions are partially ignored— is a more useful form of behavioural equivalence. Here the problem is not specific to the π-calculus semantics as there is no established domain-theoretic model for weak bisimulation even in pure CCS. (Work in this direction in the context of open maps can be found in [12].)

- *Early and open bisimulations* [21, 30] differ from the late one because of different requirements on name instantiations. Early bisimulation, which is coarser than late bisimilarity, inverts the order of the existential and universal quantifiers in the input clause (see [9] for an indication of how this can be captured using constructions on presheaf models). Open bisimilarity, which is finer than late bisimilarity and congruence, takes the open-semantic perspective into the definition of bisimulation. The only constraints on equalities among names are those imposed by name extrusion and are recorded as a *distinction* in the bisimilarity clauses. Finding a satisfactory domain equation for open bisimulation seems to be difficult, but the treatment of constraints in Section 7 might provide some hints.

REFERENCES

1. S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991.
2. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (extended abstract). In M. Hagiya and J.C. Mitchell, editors, *Proceedings, Theoretical Aspects of Computer Software. International Symposium TACS'94*, volume 789 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
3. S. Abramsky and A. Jung. Domain theory. In *Handbook of Logic in Computer Science*. Oxford University Press, 1994.
4. L. Aceto and A. Ingólfsdóttir. CPO models for compact GSOS languages. *Information and Computation*, 129(2):107–141, 1996.
5. S.O. Anderson and A.J. Power. A representable approach to finite nondeterminism. *Theoretical Computer Science* 177(1):3–25, 1997.
6. J. Baeten and W. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.
7. M. Boreale and R. De Nicola. A Symbolic Semantics for the π-calculus. *Information and Computation* 126:34–52, 1996.
8. T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*. North Holland, 1989.
9. G.L. Cattani, I. Stark and G. Winskel. Presheaf Models for the π-Calculus. In *7th International Conference on Category Theory and Computer Science*, volume 1290 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.
10. P. Cenciarelli and E. Moggi. A syntactic approach to modularity in denotational semantics. In *Proceedings, CTCS-5 (Category Theory and Computer Science Fifth Biennial Meeting)*, pages 9–12. CWI, September 1993.
11. M.P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Cambridge University Press Distinguished Dissertations in Computer Science, 1996.
12. M.P. Fiore, G.L. Cattani and G. Winskel. Weak Bisimulation and Open Maps. In *Fourteenth LICS Conf.* IEEE Computer Society Press, 1999.
13. M. Hennessy. A term model for synchronous processes. *Information and Control*, 51(1):58–75, October 1981.
14. M. Hennessy. A fully abstract denotational model for higher-order processes. *Information and Computation*, 112(1):55–95, July 1994.
15. M. Hennessy. A fully abstract denotational semantics for the pi-calculus. Computer Science Technical Report 1996:04, School of Cognitive and Computing Sciences, University of Sussex, 1996.

16. M. Hennessy and G. Plotkin. Full abstraction for a simple parallel programming language. In J. Becvar, editor, *Proceedings, 8th Symposium on Mathematical Foundations of Computer Science*, volume 74 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

17. M. Hennessy and G. Plotkin. A term model for CCS. In P. Dembinski, editor, *Proceedings, 9th Symposium on Mathematical Foundations of Computer Science*, volume 88 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

18. J.M.E. Hyland and C.-H.L. Ong. Pi-calculus, dialogue games and PCF. In *Proceedings, 7th Annual ACM Conference on Functional Programming Languages and Computer Architecture*. ACM, 1995.

19. A. Jeffrey. A fully abstract semantics for a concurrent functional language with monadic types. In *Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science*, pages 255–264. IEEE Computer Society Press, 1995.

20. J. Lambek and P.J. Scott. Introduction to Higher Order Categorical Logic. Cambridge University Press, 1986.

21. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, September 1992.

22. R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, University of Edinburgh, 1991. (Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.)

23. E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, July 1991.

24. F.J. Oles. Type algebras, functor categories and block structure. In M. Nivat and J.C. Reynolds, editors, *Algebraic Methods in Semantics*, 1985.

25. J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Information and Computation*, 120(2):174–197, 1 August 1995.

26. A.M. Pitts and I.D.B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In A.M. Borzyszkowski and S. Sokolowski, editors, *Proceedings, MFCS'93*, volume 711 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

27. J. Rutten. Processes as terms: non-well-founded models for bisimulation. *Mathematical Structures in Computer Science*, 2:257–275, 1992.

28. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, Department of Computer Science, University of Edinburgh, 1992.

29. D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

30. D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996.

31. D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–274, 1996.

32. D. Sangiorgi and D. Walker. *The π-calculus: a theory of mobile processes.* Cambridge University Press. Forthcoming.

33. M. Smyth and G. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11(4):761–783, November 1982.

34. I. Stark. A fully abstract domain model for the π-calculus. In *Eleventh LICS Conf.* IEEE Computer Society Press, 1996.

35. B. Thomsen. *Calculi for Higher Order Communicating Systems.* PhD thesis, Department of Computing, Imperial College, 1990.

APPENDIX A

Denotational interpretations

A.1. OPEN INTERPRETATION

We give a complete definition of the open interpretation hinted at in Subsection 4.6.

The open interpretation of a name $a \in V$ and a process $P \in \mathcal{P}r_V$, respectively denoted $\mathcal{O}[\![V \vdash a]\!] : N^{|V|} \dot{\to} N$ and $\mathcal{O}[\![V \vdash P]\!] : N^{|V|} \dot{\to} A$, are defined as follows.

- $\mathcal{O}[\![a_1, \ldots, a_n \vdash a_i]\!] \overset{\text{def}}{=} \pi_i$ $(1 \leq i \leq n)$.
- $\mathcal{O}[\![V \vdash a(b).\, P]\!] \overset{\text{def}}{=} \mathsf{in} \circ \langle \mathcal{O}[\![V \vdash a]\!], \lambda\, \mathcal{O}[\![V, b \vdash P]\!] \rangle$.
- $\mathcal{O}[\![V \vdash \overline{a}b.\, P]\!] \overset{\text{def}}{=} \mathsf{out} \circ \langle \mathcal{O}[\![V \vdash a]\!], \mathcal{O}[\![V \vdash b]\!], \mathcal{O}[\![V \vdash P]\!] \rangle$.
- $\mathcal{O}[\![V \vdash \overline{a}(b).\, P]\!] \overset{\text{def}}{=} \mathsf{bout} \circ \langle \mathcal{O}[\![V \vdash a]\!], \lambda\, \mathcal{O}[\![V, b \vdash P]\!] \rangle$
- $\mathcal{O}[\![V \vdash \tau.\, P]\!] \overset{\text{def}}{=} \mathsf{tau} \circ \mathcal{O}[\![V \vdash P]\!]$.
- $\mathcal{O}[\![V \vdash \mathbf{0}]\!] \overset{\text{def}}{=} \lambda v.\, \mathsf{nil}$.
- $\mathcal{O}[\![V \vdash P + Q]\!] \overset{\text{def}}{=} \mathsf{sum} \circ \langle \mathcal{O}[\![V \vdash P]\!], \mathcal{O}[\![V \vdash Q]\!] \rangle$.
- $\mathcal{O}[\![V \vdash P \mid Q]\!] \overset{\text{def}}{=} \mathsf{par} \circ \langle \mathcal{O}[\![V \vdash P]\!], \mathcal{O}[\![V \vdash Q]\!] \rangle$.
- $\mathcal{O}[\![V \vdash P \lfloor\!\lfloor Q]\!] \overset{\text{def}}{=} \mathsf{lm} \circ \langle \mathcal{O}[\![V \vdash P]\!], \mathcal{O}[\![V \vdash Q]\!] \rangle$.
- $\mathcal{O}[\![V \vdash P \parallel Q]\!] \overset{\text{def}}{=} \mathsf{syn} \circ \langle \mathcal{O}[\![V \vdash P]\!], \mathcal{O}[\![V \vdash Q]\!] \rangle$.
- $\mathcal{O}[\![V \vdash \boldsymbol{\nu} a\, P]\!] \overset{\text{def}}{=} \mathsf{res} \circ \lambda\, \mathcal{O}[\![V, a \vdash P]\!]$.
- $\mathcal{O}[\![V \vdash [a = b]P]\!] \overset{\text{def}}{=} \mathsf{M} \circ \langle \mathcal{O}[\![V \vdash a]\!], \mathcal{O}[\![V \vdash b]\!], \mathcal{O}[\![V \vdash P]\!] \rangle$.
- $\mathcal{O}[\![V \vdash [a \neq b]P]\!] \overset{\text{def}}{=} \mathsf{MM} \circ \langle \mathcal{O}[\![V \vdash a]\!], \mathcal{O}[\![V \vdash b]\!], \mathcal{O}[\![V \vdash P]\!] \rangle$.
- $\mathcal{O}[\![V \vdash !\, a(b).\, P]\!] \overset{\text{def}}{=} \mathsf{!in} \circ \langle \mathcal{O}[\![V \vdash a]\!], \lambda\, \mathcal{O}[\![V, b \vdash P]\!] \rangle$.
- $\mathcal{O}[\![V \vdash !\, \overline{a}b.\, P]\!] \overset{\text{def}}{=} \mathsf{!out} \circ \langle \mathcal{O}[\![V \vdash a]\!], \mathcal{O}[\![V \vdash b]\!], \mathcal{O}[\![V \vdash P]\!] \rangle$.
- $\mathcal{O}[\![V \vdash !\, \overline{a}(b).\, P]\!] \overset{\text{def}}{=} \mathsf{!bout} \circ \langle \mathcal{O}[\![V \vdash a]\!], \lambda\, \mathcal{O}[\![V, b \vdash P]\!] \rangle$.
- $\mathcal{O}[\![V \vdash !\, \tau.\, P]\!] \overset{\text{def}}{=} \mathsf{!tau} \circ \mathcal{O}[\![V \vdash P]\!]$.

As a *standard* lemma we have:

Lemma A.1.

1.*(Alpha-conversion) Let $V \vdash P$ and $V \vdash Q$. For P and Q alpha-convertible,*

$$\mathcal{O}[\![V \vdash P]\!] = \mathcal{O}[\![V \vdash Q]\!].$$

2.*(Permutation) Let $V, a, b, W \vdash P$. Then,*

$$\mathcal{O}[\![V, a, b, W \vdash P]\!] = \mathcal{O}[\![V, b, a, W \vdash P]\!] \circ \langle \pi_1, \ldots, \pi_{|V|}, \pi_{|V|+2}, \pi_{|V|+1} \pi_{|V|+3}, \ldots, \pi_{|V|+|W|+2} \rangle.$$

3.*(Contraction) Let $V, W \vdash P$. For $a \notin V, W$,*

$$\mathcal{O}[\![V, a, W \vdash P]\!] = \mathcal{O}[\![V, W \vdash P]\!] \circ \langle \pi_1, \ldots, \pi_{|V|}, \pi_{|V|+2}, \ldots, \pi_{|V|+|W|+1} \rangle.$$

4.*(Substitution) Let $V, a, W \vdash P$. For $b \in V, W$,*

$$\mathcal{O}[\![V, W \vdash P\{{}^{b}\!/\!{}_{a}\}]\!] = \mathcal{O}[\![V, a, W \vdash P]\!] \circ \langle \pi_1, \ldots, \pi_{|V|}, \mathcal{O}[\![V, W \vdash b]\!], \pi_{|V|+1}, \ldots, \pi_{|V|+|W|} \rangle.$$

COROLLARY A.1 (Invariance under injective substitutions)). *Let $V, a, W \vdash P$. For $b \notin V, W$,*

$$\mathcal{O}[\![V, a, W \vdash P]\!] = \mathcal{O}[\![V, b, W \vdash P\{b/a\}]\!].$$

A.2. QUASI-COMPOSITIONALITY OF THE CLOSED INTERPRETATION

This subsection is devoted to proving Lemma 4.1. We start by proving the following:

PROPOSITION A.1. *For $V \equiv a_0, \ldots, a_{n-1}$ we have:*

1.$(\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n(0, \ldots, n-1) = (\langle \mathcal{C}[\![V \vdash P\{a_i/b\}]\!] \rangle_{i \in n}, \mathcal{C}[\![V, b \vdash P]\!])$.
2.$\delta_n\ ((\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\ (0, \ldots, n-1))\ \mathsf{new}_n = \mathcal{C}[\![V, b \vdash P]\!]$.

Proof. We write $\vec{n}$ for $(0, \ldots, n-1)$.

(1) First observe that

$$\begin{aligned}(\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\ \vec{n} &= (\lambda\ (\mathcal{O}[\![V, b \vdash P]\!]_n)\ \vec{n}, \mathcal{O}[\![V, b \vdash P]\!]_{n+1}(\vec{n}, n)) \\ &\qquad \text{, by Proposition 4.2} \\ &= (\lambda\ (\mathcal{O}[\![V, b \vdash P]\!]_n)\ \vec{n}, \mathcal{C}[\![V, b \vdash P]\!]).\end{aligned}$$

Moreover, since for $1 \leq i \leq n$,

$$\begin{aligned}\mathcal{C}[\![V \vdash P\{a_i/b\}]\!] &= \mathcal{O}[\![V \vdash P\{a_i/b\}]\!]_n\ \vec{n} = \mathcal{O}[\![V, b \vdash P]\!]_n(\vec{n}, \mathcal{O}[\![V \vdash a_i]\!]_n\ \vec{n}) \\ &= \mathcal{O}[\![V, b \vdash P]\!]_n(\vec{n}, i)\end{aligned}$$

it follows that

$$\lambda\ (\mathcal{O}[\![V, b \vdash P]\!]_n)\ \vec{n}\ i = \mathcal{C}[\![V \vdash P\{a_i/b\}]\!]$$

and we are done.

(2) Since

- $\delta_n : (N \Rightarrow X)(n) \to (\delta N \Rightarrow \delta X)(n)$ is given by

$$X(n)^n \times X(n+1) \xrightarrow{X(\iota_n)^n \times \langle X(\iota_n + 1), \mathrm{id} \rangle} X(n+1)^n \times X(n+2) \times X(n+1)$$

where ι_n denotes the inclusion $n \subseteq n+1$; and,
- for $h : \delta N \Rightarrow X$, we have $h_n : X(n)^n \times X(n+1) \times X(n)$ and

$$h_n(\mathsf{new}_n) = \mathsf{Eval}_n(h_n, n) = \pi_3(h_n) : X(n),$$

it follows that

$$\begin{aligned}\delta_n\ ((\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\ \vec{n})\ \mathsf{new}_n &= \pi_3\ (\delta_n\ ((\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\ \vec{n})) \\ &= \pi_2\ ((\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\ \vec{n}) \\ &= \mathcal{C}[\![V, b \vdash P]\!].\end{aligned}$$

■

PROOF OF LEMMA 4.1: We only consider the interesting cases:

(2) Let $| V |= n$ and write $\vec{n}$ for $(0,\dots,n-1)$. Then,

$$\begin{aligned}\mathcal{C}[\![V \vdash \overline{a}(b).\,P]\!] &= \mathcal{O}[\![V \vdash \overline{a}(b).\,P]\!]_n\,\vec{n}\\ &= \mathsf{bout}_n(\mathcal{O}[\![V \vdash a]\!]_n\,\vec{n}, (\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\,\vec{n})\\ &= (\mathsf{S}_3)_n(\mathcal{C}[\![V \vdash a]\!], \delta_n\ ((\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\,\vec{n})\ \mathsf{new}_n)\end{aligned}$$

and, by Proposition A.1 (2), we are done.

(3) Writing $\vec{n}$ for $(0,\dots,n-1)$,

$$\begin{aligned}\mathcal{C}[\![V \vdash a(b).\,P]\!] &= \mathcal{O}[\![V \vdash a(b).\,P]\!]_n\,\vec{n}\\ &= \mathsf{in}_n(\mathcal{O}[\![V \vdash a]\!]_n\vec{n}, (\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\,\vec{n})\\ &= \mathsf{in}_n(\mathcal{C}[\![V \vdash a]\!], (\lambda\ \mathcal{O}[\![V, b \vdash P]\!])_n\,\vec{n}),\end{aligned}$$

and, by Proposition A.1 (1), we are done.

(8) Let $| V |= n$ and write $\vec{n}$ for $(0,\dots,n-1)$. Then,

$$\begin{aligned}\mathcal{C}[\![V \vdash \boldsymbol{\nu} a\, P]\!] &= \mathcal{O}[\![V \vdash \boldsymbol{\nu} a\, P]\!]_n\vec{n}\\ &= \mathsf{R}_n(\delta_n\ ((\lambda\ \mathcal{O}[\![V, a \vdash P]\!])_n\vec{n})\ \mathsf{new}_n)\end{aligned}$$

and, by Proposition A.1 (2), we are done. ■

APPENDIX B
Some category-theoretic notions

DEFINITION B.1. (Algebra for an endofunctor) For a (strong) endofunctor F, a map $FA \to A$ is said to be an *F-algebra*. Moreover, a map $h : A \to B$ is said to be an *F-homomorphism* from $a : FA \to A$ to $b : FB \to B$ iff $b \circ Fh = h \circ a : FA \to B$.

DEFINITION B.2. (Algebra for a monad) For a (strong) monad T, a map $a : TA \to A$ is a *T-algebra* iff the diagram

$$\begin{tikzcd} & TA \arrow[d, "a"] & T^2A \arrow[l, "Ta"'] \arrow[d, "\mu^T_A"] \\ A \arrow[ur, "\eta^T_A"] \arrow[r, "\mathrm{id}_A"'] & A & TA \arrow[l, "a"] \end{tikzcd}$$

commutes. Moreover, a map $h : A \to B$ is said to be a *T-homomorphism* from $a : TA \to A$ to $b : TB \to B$ iff $b \circ Th = h \circ a : TA \to B$.

Definition B.3. (Monad morphism) For (strong) monads S and T on the same category, a natural transformation $\lambda : S \dot{\to} T$ is said to be a *monad morphism* from S to T iff the diagram

$$\begin{tikzcd}
 & SX \arrow[d, "\lambda_X"] & S^2X \arrow[l, "\mu^S_X"'] \arrow[r, "\lambda_{SX}"'] & T(SX) \arrow[dl, "T(\lambda_X)"] \\
X \arrow[ur, "\eta^S_X"] \arrow[r, "\eta^T_X"'] & TX & T^2X \arrow[l, "\mu^T_X"] &
\end{tikzcd}$$

commutes.

Definition B.4. (Distributive law) For a (strong) monad T and a (strong) endofunctor F on the same category, the natural transformation $\sigma : FT \dot{\to} TF$ is said to be a *distributive law* iff the diagram

$$\begin{tikzcd}
 & F(TX) \arrow[d, "\sigma_X"] & F(T^2X) \arrow[l, "F(\mu^T_X)"'] \arrow[r, "\sigma_{TX}"'] & T(F(TX)) \arrow[dl, "T(\sigma_X)"] \\
FX \arrow[ur, "F(\eta^T_X)"] \arrow[r, "\eta^T_{FX}"'] & T(FX) & T^2(FX) \arrow[l, "\mu^T_{FX}"] &
\end{tikzcd}$$

commutes.