Under consideration for publication in Math. Struct. in Comp. Science

The λ-calculus in the π-calculus

Xiaojuan Cai[†] Yuxi Fu[‡]

BASICS, Department of Computer Science

Shanghai Jiaotong University, Shanghai 200240

MOE-MS Key Laboratory for Intelligent Computing and Intelligent Systems

Received 16 December 2009; Revised 19 January 2011

A general approach is proposed that transforms, in the framework of pi calculus, objects to methods in an on-the-fly manner. The power of the approach is demonstrated by applying it to generate an encoding of the full lambda calculus in the pi calculus. The encoding is proved to preserve and reflect beta reduction and is shown to be fully abstract with respect to Abramsky's applicative bisimilarity.

Keywords. Process calculus, the λ-calculus, the π-calculus, encoding

Contents

1 Introduction 2
2 Technical Background 5
2.1 Lambda Calculus 5
2.2 Pi Calculus 7
2.3 Milner's Translation 10
3 Data Structure Defined in Pi 11
3.1 Tree Structure 12
3.2 Operation on Tree 13
3.3 Turning Object into Method 14
3.4 Operation in Tree 16
4 Encoding in Pi 16
4.1 Structural Tree for λ-Term 17
4.2 Encoding of λ-Term 17
4.3 Simulation of β-Reduction 20

[†] cxj@sjtu.edu.cn
[‡] fu-yx@cs.sjtu.edu.cn

5 Property of Encoding 21
 5.1 Reduction Action 23
 5.2 Input Action 26
 5.3 Correspondence Property 29
6 Correctness of Encoding 33
 6.1 Operational Soundness and Completeness 33
 6.2 Full Abstraction 34
7 Conclusion 38
References 40
Appendix A Proofs of Section 5.3 43
 A.1 Proofs of Lemma 4 and Lemma 5 43
 A.2 Proof of Lemma 6 52
 A.3 Proof of Lemma 7 54

1. Introduction

The π-calculus of Milner, Parrow and Walker (1992) has been successful in both theory and practice. As a theoretical model, it has been shown that name passing communication mechanism is both stable and powerful in that many variants of the π-calculus turn out to be just as powerful (Nestmann and Pierce 2000; Palamidessi 2003; Fu and Lu 2010). The investigations have also shown that the π-calculus captures much of the expressiveness of mobile computing. On the practical side, the π-calculus acts as a ubiquitous model to interpret various phenomena in mobile and distributed computing. Studies have indicated that the π-calculus is extremely useful in programming objects, methods, specifications and protocols widely used in modern computing.

Two issues concerning the pragmatics of the π-calculus is particularly interesting. One is how the objects of a source model/calculus/language are interpreted and what kind of features are captured. The other is what particular variant of the π-calculus is used as a target model in a particular application. In what follows we inspect these two aspects by taking a look at some of the previous works.

Walker's encoding of a parallel object oriented language (Walker 1991; Walker 1995) is carried out in the minimal π-calculus that has neither conditionals nor any forms of choice operation. This interpretation is natural since the naming policy of the π-calculus goes along with the idea of object oriented paradigm very well. The reference of an object and the invoking of a method are interactions and passing of references is simply passing of names. Amadio and Prasad's specification of the Mobile IP protocol (Amadio and Prasad 2000) makes use of the asynchronous π-calculus (Boudol 1992; Honda and Tokoro 1991a; Honda and Tokoro 1991b) with choice, match condition and parametric definition. The choice and the condition operators strictly enhance the expressive power of the π-calculus (Fu and Lu 2010). Parametric definition is normally strictly more powerful than fixpoint operator or replication operator (Sangiorgi 1996). Due to the fact that the π-calculus in (Amadio and Prasad 2000) is mainly used as a description language that specifies protocols, the localization operator is absent. The description power of the π-calculus has also been explored to specify security protocols. Abadi and Gordon

(1999) extend the π-calculus to Spi in an effort to deal with security issues. Baldamus, Parrow and Victor (2004) provide an interesting translation of Spi into the π-calculus with guarded choice and match. The basic idea of the translation is to understand the terms of Spi as objects. The ability of the π-calculus to dynamically create local names is essential for the interpretation of encryption and decryption.

A computation is typically composed of data flow and control flow. The philosophy of the first order π-calculus is that everything is seen as control flow management. To encode in the π-calculus is to think in terms of access (name) control. But how is this new paradigm compared to the traditional ones in terms of expressiveness? We have already mentioned how object-oriented programming fits in the framework of the π-calculus. The functional paradigm and the higher order feature can also be realized by control flow management. Thomsen (1993; 1995) discusses the issue of how to translate higher order CCS in the π-calculus. Sangiorgi (1993a; 1993b) proves that the higher order π-calculus is equivalent to the first order π-calculus in a strong sense. Other researches confirm that the π-calculus is more or less equivalent to many of its variants (Palamidessi 2003; Fu and Lu 2010). These results add considerable weight to the authority of the π-calculus.

All these works and many others have suggested that encoding in the π-calculus is very much like programming. Like in all familiar programming paradigms, programming in the π-calculus has a certain methodology, the π-methodology. This methodology provides a general guideline on what the entities are modeled, how the computations are simulated and which equalities are respected.

- In π-programming, there are two kinds of entities. The objects are simply π-processes, possibly with distinguished reference names. The methods are replicated forms of prefixed processes. A method can be invoked as often as necessary. Terms of a source language are normally interpreted as objects in the π-calculus, whereas types are typically interpreted as methods.
- In π-programs, computations are nothing but reference management, which controls how names should and should not be passed around. The computational behavior of a term is defined by the operators and the structure of the term. The structural aspect of source language is captured in π-programs by suitable data structures, which are either explicit or implicit. Simulations of computations of source language are achieved by reference management and are organized through data structures.
- The intensional equalities of source models are interpreted by the observational equivalence of the π-calculus. The quality of an interpretation is judged by a full abstraction property. The existence of such a property is more likely if the intensional equality is observational. It is difficult for the observational theory of the π-calculus to capture the intensional equality if the latter is defined by some extra-logical theory.

Programming in the π-calculus can be greatly simplified by having a richer set of operators. For example the guarded choice and the match operator are always convenient. However in a particular application, the use of such operators may not be indispensable. A useful strategy is to code up source language in some super model of the minimal π-calculus to start with, and then look for the way to transplant the encoding to a sub-calculus. For a source calculus it is interesting to investigate a minimal set of operators

necessary to carry out a translation into the π-calculus. A result of this kind says a great deal about the nature of the source model.

Milner's interpretation (Milner 1992) of the lazy λ-calculus (Abramsky 1990) is an instructive example of using the general encoding strategy outlined above. An abstraction λ-term $\lambda x.M$ is coded up by an object with a special name, say 'λ'. The encoding of an application term MN has an underlying data structure of a binary tree, with two children being the encodings of M and N respectively. A variable x is translated to a call, a leaf in the binary tree, that may invoke a method named x. Milner shows that his encoding preserves β-conversion. Sangiorgi (1993a; 1995) strengthens the result by a full abstraction theorem, stating that the observational equivalence induced by the encoding is precisely the open applicative bisimilarity, a generalization of Abramsky's applicative bisimilarity (Abramsky 1990) to open λ-terms. By modifying the reference management of this encoding, Milner also provides in (Milner 1992) an encoding of the call-by-value λ-calculus (Plotkin 1975). After Milner's work, his encodings have been studied in several different settings. For example Sangiorgi (1993b) points out that the close relationship between the lazy λ-calculus (the call-by-name λ-calculus) and the higher order π-calculus can be explored to generate encodings in the first order π-calculus, using the encoding from the higher order π-calculus to the first order π-calculus. The encodings of the strong call-by-name λ-calculus are studied in several variants of the π-calculus (Fu 1997; Parrow and Victor 1997; Fu 1999; Merro 2004).

Despite of all the efforts, no encoding of the full λ-calculus has been proposed. One expects that such an encoding satisfies at least two properties. One is that it should preserve and reflect the operational semantics of the full λ-calculus. The other is that the encoding should render true a full abstraction theorem with respect to some observational equivalence on the λ-terms. The difficulty of coding up the full λ-calculus in the π-calculus is discussed in (Milner 1992). It is pointed out by Milner that, from the point of view of the π-calculus, the full λ-reduction strategy appears odd. In the following reduction sequence

$$\begin{array}{rcl} (\lambda x.M)N & \rightarrow^* & (\lambda x.M')N' \\ & \rightarrow & M'\{N'/x\} \\ & \rightarrow^* & \ldots N'_1 \ldots N'_2 \ldots \end{array}$$

the β-reduction $(\lambda x.M')N' \rightarrow M'\{N'/x\}$ is a turning point. Before it happens, the term N evolves by itself. After the β-reduction two copies of the descendant N' of N may evolve independently in parallel. If N is modeled by a method, then it is underneath a replication operator. But then one could hardly explain $N \rightarrow^* N'$. If N is interpreted as an object, then it seems difficult to model the cloning of N' into several copies. The idea of interpreting $M'\{N'/x\}$ by syntactical substitution does not work either. To model the cloning by a higher order process, N has to appear in a higher order output prefix, which makes an interpretation of $N \rightarrow^* N'$ and $\lambda x.M \rightarrow^* \lambda x.M'$ very unlikely. These difficulties suggest that there is probably something more about π-programming than we have understood.

The design of an operationally sound and complete, and observationally fully abstract

encoding of the full λ-calculus in the π-calculus has been an open issue for nearly twenty years. This problem is resolved in this paper.

The structure of the paper is as follows: Section 2 lays down the necessary background technicalities. Section 3 explains how to define data structures in the π-calculus. Section 4 defines the encoding of the full λ-calculus in the π-calculus. Section 5 discusses properties of the encoding. Section 6 establishes the correctness of the encoding. Section 7 concludes with a remark on further work. All the proofs of the lemmas stated in Section 6 are placed in Appendix A.

2. Technical Background

This section covers the necessary background materials on the λ-calculus, the π-calculus, and Milner's encoding. In sequel, $\{\diamond_0/\circ_0, \ldots, \diamond_n/\circ_n\}$ denotes a substitution that replaces $\circ_0, \ldots, \circ_n$, necessarily pairwise distinct, by $\diamond_0, \ldots, \diamond_n$ respectively.

2.1. *Lambda Calculus*

Church's λ-calculus (Barendregt 1984) is an operational model of the functional computation. The entities of the model are λ-terms and the computations are β-reductions. Let $\mathcal{V}$ be the set of the term variables, ranged over by $x, y, z, \ldots$. The set Λ of the λ-terms is defined by the following grammar:

$$M := x \mid \lambda x.M \mid MM'.$$

The variable x in $\lambda x.M$ is bound. A variable is free if it is not bound. A λ-term is closed if it does not contain any free variables. Let Λ^0 be the set of the closed λ-terms. Let $\rightarrow^*$ be the reflexive and transitive closure of $\rightarrow$. The λ-contexts are defined by the following grammar:

$$C[_] ::= _ \mid \lambda x.C[_] \mid (C[_])M \mid M(C[_]),$$

where M is a λ-term. We say that $C[_]$ is a closing λ-context for M if $C[M] \in \Lambda^0$. The reduction strategy of the λ-terms is defined by the following rules:

$$\frac{}{(\lambda x.M)N \rightarrow M\{N/x\}}\ \beta \text{ reduction} \qquad \frac{M \rightarrow M'}{MN \rightarrow M'N}\ \text{structural rule}$$

$$\frac{N \rightarrow N'}{MN \rightarrow MN'}\ \text{eager evaluation} \qquad \frac{M \rightarrow M'}{\lambda x.M \rightarrow \lambda x.M'}\ \text{partial evaluation}$$

Using the notion of λ-context, the four reduction rules can be replaced by the following single reduction rule:

$$\frac{}{C[(\lambda x.M)N] \rightarrow C[M\{N/x\}]}\ C[_] \text{ is a } \lambda\text{-context.}$$

The intensional equality on the λ-terms is the well-known β-conversion $=_\beta$, the equality generated by $\rightarrow$. From the observational viewpoint, $=_\beta$ is too fine-grained. Take for instance the closed term $\Omega \stackrel{\text{def}}{=} (\lambda x.xx)(\lambda x.xx)$. One has that $\Omega \neq_\beta \Omega\Omega$. But clearly $\Omega\Omega$

is just as solitary as Ω. The importance of the observational theory of the λ-calculus was emphasized by a number of people. Abramsky (1990) for example introduces the applicative bisimilarity and develops a coherent observational theory for a particular reduction strategy, the lazy λ-reduction strategy. The observational approach is of course applicable to other reduction strategies of the λ-calculus. The following definition of applicative bisimilarity is due to Abramsky.

Definition 1. A binary relation $\mathcal{R}$ on Λ^0 is an applicative bisimulation if the following properties hold whenever $M\mathcal{R}N$.
(i) If $M \to^* \lambda x.M'$ then $N \to^* \lambda x.N'$ for some N' such that $M'\{L/x\} \ \mathcal{R} \ N'\{L/x\}$ for every $L \in \Lambda^0$.
(ii) If $N \to^* \lambda x.N'$ then $M \to^* \lambda x.M'$ for some M' such that $M'\{L/x\} \ \mathcal{R} \ N'\{L/x\}$ for every $L \in \Lambda^0$.
The applicative bisimilarity $=_a$ is the largest applicative bisimulation.

The relation $=_a$ is obviously an equivalence. For λ-terms M, N containing say the free variable x, we could define $M =_a N$ iff $\lambda x.M =_a \lambda x.N$. Then $=_a$ becomes a congruence on the set of all λ-terms.

Lemma 1. If $M \to M'$ then $M =_a M'$.

Proof. If $M' \to^* \lambda x.M''$ then $M \to^* \lambda x.M''$. If $M \to^* \lambda x.M''$ then by Church-Rosser property there exists some M''' such that $M' \to^* M'''$ and $\lambda x.M'' \to^* M'''$. Clearly M''' must be of the form $\lambda x.M''''$. Therefore

$$\{(M, M') \mid \Lambda^0 \ni M \to^* M'\}$$

is an applicative bisimulation. $\square$

It follows that beta conversion $=_\beta$ is included in $=_a$. The inclusion is strict since $\Omega \neq_\beta \Omega\Omega$ but $\Omega =_a \Omega\Omega$. Another concrete example of Lemma 1 is $(\lambda y.\mathbf{I})\Omega =_a \mathbf{I}$, where $\mathbf{I} \equiv \lambda x.x$. What it tells us is that the applicative bisimilarity, and the β-conversion as well, is not termination preserving. This is in contrast to the situation in the lazy λ-calculus. One implication of this fact is that, as long as we take the applicative bisimilarity as the equality on the λ-terms, the equivalence of the π-calculus should not be termination preserving either, or a full abstraction result would be impossible. An alternative would be to refine $=_a$ so that termination is taken into consideration. This would imply an overhaul of the theory of the λ-calculus since β-conversion would no longer be valid. In this paper we shall stick to Definition 1.

An alternative characterization of the applicative bisimilarity is given by the next lemma.

Lemma 2. The applicative bisimilarity $=_a$ is the largest among all $\mathcal{R}$ that satisfies the following properties:

1 $\mathcal{R}$ is symmetric.
2 If $M\mathcal{R}N \to N'$ then $M\mathcal{R}N'$.
3 If $(\lambda x.M)\mathcal{R}N$ then $N \to^* \lambda x.N'$ for some N' such that $(\lambda x.M)\mathcal{R}(\lambda x.N')$.

4 If $(\lambda x.M)\mathcal{R}(\lambda x.N)$ then $M\{L/x\}\mathcal{R}N\{L/x\}$ for every $L \in \Lambda^0$.

Proof. Since $M \to M'$ implies $M =_a M'$, the applicative bisimilarity satisfies the above properties. Conversely a relation satisfying these properties is a subset of $=_a$. $\square$

2.2. *Pi Calculus*

We assume that there is a set $\mathcal{N}$ of names and a set $\mathcal{N}_v$ of name variables. The following conventions will be enforced throughout the paper:

— The set $\mathcal{N}$ is ranged over by a, b, c, d;
— The union set $\mathcal{N} \cup \mathcal{N}_v$ is ranged over by $e, f, g, \ldots, x, y, z$.

So it is syntactically incorrect to write for example $a(c).T$. Moreover the following additional assumption will be strictly respected:

If we write for example $(n)T$ then the explicit n in $(n)T$ should be understood as a name. Similarly if we write $a(n).T$ then n in $a(n).T$ must be understood as a name variable.

It will become clear that our conventions are very convenient when defining the structural encoding of the λ-calculus in the π-calculus. We write $\widetilde{n}$ for a finite set of names/name variables $\{n_0, n_1, \cdots, n_i\}$ and we also abbreviate $(n_0 n_1 \cdots n_i)T$ to $(\widetilde{n})T$. When $\widetilde{n}$ is empty, $(\widetilde{n})T$ is just T.

The π-calculus (Milner, Parrow and Walker 1992) has a number of variants. The common part of all these variants is the minimal π-calculus, denoted by π^M, which is the variant used by Milner to encode the lazy λ-calculus. The grammar of the π^M-terms is given by the following BNF.

$$R, S, T, \ldots \quad := \quad \pi.T \mid S \,|\, T \mid (a)T \mid !T,$$

where

$$\pi \quad := \quad \tau \mid n(x) \mid \overline{n}m.$$

The prefix π may be an internal action, an input $a(x)$ or an output $\overline{n}m$. In $a(x).T$ the name variable x is bound. A name variable is free if it is not bound. The name a in $(a)T$ is a local name. A name is global if it is not local. The bounded output prefix term $\overline{n}(c).T$ is defined by $(c)\overline{n}c.T$. We will write A, B, C, D, O, P, Q for the processes, which are terms without any free name variables. The operational semantics is defined by the following rules, in which μ ranges over the set $\mathcal{A} = \{ab, \overline{a}b, \overline{a}(b) \mid a, b \in \mathcal{N}\} \cup \{\tau\}$ of the action labels.

Action

$$\frac{}{\tau.T \xrightarrow{\tau} T} \qquad \frac{}{\overline{a}b.T \xrightarrow{\overline{a}b} T} \qquad \frac{}{a(x).T \xrightarrow{ab} T\{b/x\}}$$

Composition

$$\frac{S \xrightarrow{\mu} S'}{S \,|\, T \xrightarrow{\mu} S' \,|\, T} \qquad \frac{S \xrightarrow{ab} S' \quad T \xrightarrow{\overline{a}b} T'}{S \,|\, T \xrightarrow{\tau} S' \,|\, T'} \qquad \frac{S \xrightarrow{ab} S' \quad T \xrightarrow{\overline{a}(b)} T'}{S \,|\, T \xrightarrow{\tau} (b)(S' \,|\, T')}$$

Localization

$$\frac{T \xrightarrow{\overline{a}b} T'}{(b)T \xrightarrow{\overline{a}(b)} T'}\ a, b \text{ are distinct.} \qquad \frac{T \xrightarrow{\mu} T'}{(b)T \xrightarrow{\mu} (b)T'}\ b \text{ is not in } \mu.$$

Recursion

$$\frac{T \,|\, !T \xrightarrow{\mu} T'}{!T \xrightarrow{\mu} T'}$$

In the first composition rule, μ should not contain any global name in T.

The minimal π-calculus turns out to be surprisingly powerful. It is however not very convenient when it comes down to programming. Moreover it is not yet clear if the full abstraction result (Theorem 2) is achievable if the target model is π^M. In this paper we shall work with π^{def}, the π-calculus with parametric definition and guarded choice. The set of the π^{def}-terms is generated from the following grammar:

$$R, S, T, \ldots \quad := \quad \sum_{i \in I} \varphi_i \pi_i.T_i \mid S \,|\, T \mid (a)T \mid D(\widetilde{x}),$$

where I is a finite indexing set and φ_i is a condition. We write $\mathcal{P}$ for the set of the π^{def}-processes. A condition is defined by the following grammar (Parrow and Sangiorgi 1995):

$$\varphi, \psi \quad := \quad \top \mid \bot \mid p = q \mid p \neq q \mid \varphi \wedge \psi,$$

where $p = q$ is a match and $p \neq q$ is a mismatch, $\top$ and $\bot$ are respectively the logical truth and false. The conjunction $\varphi \wedge \psi$ is often abbreviated to the concatenation $\varphi\psi$. The semantics of *case term* $\sum_{i \in I} \varphi_i \pi_i.T_i$ is given by the following rules, in which $\Leftrightarrow$ stands for the logical equivalence.

$$\frac{}{\sum_{i \in I} \varphi_i \pi_i.T_i \xrightarrow{\pi_i} T_i}\ \pi_i \text{ is an output or } \tau, \text{ and } \varphi_i \Leftrightarrow \top.$$

$$\frac{}{\sum_{i \in I} \varphi_i \pi_i.T_i \xrightarrow{ab} T_i\{b/x\}}\ \pi_i = a(x) \text{ and } \varphi_i \Leftrightarrow \top.$$

We will use extended choices of the form $\sum_{i=1..n} \varphi_i \mu_i.T_i$, where the prefixes could be bounded output prefixes. We often write $\varphi_1\mu_1.T_1 + \ldots + \varphi_n\mu_n.T_n$ for $\sum_{i=1..n} \varphi_i \mu_i.T_i$. Another alternative notation is

begin case
$\varphi_1 \Rightarrow \mu_1.T_1$
$\vdots$
$\varphi_n \Rightarrow \mu_n.T_n$
end case.

The well-known "**if_then_else**" can be defined in terms of the case terms. For example

$$\textbf{if } p{=}q \wedge u{\neq}v \textbf{ then } \mu_1.T_1 \textbf{ else } \mu_2.T_2$$

is defined by the term

$$(p{=}q \wedge u{\neq}v)\mu_1.T_1 + (p{\neq}q)\mu_2.T_2 + (u{=}v)\mu_2.T_2.$$

Occasionally we mix several notations.

A parametric definition is given by a finite set of equations in the following form:

$$\begin{array}{rcl} D_1(\widetilde{x_1}) & = & T_1, \\ & \vdots & \\ D_n(\widetilde{x_n}) & = & T_n, \end{array}$$

where for each $i \in \{1, \ldots, n\}$ the free names of T_i must appear in $\widetilde{x}_i$. An instantiation $D_i(\widetilde{n}_i)$ of $D_i(\widetilde{x}_i)$ is the term $T_i\{\widetilde{n}_i/\widetilde{x}_i\}$. In the above definition, $D_i(\widetilde{x}_i)$ may have instantiated occurrence in any of $T_1, \ldots, T_n$. The rule that defines the operational semantics of the parametric definition is the following one.

$$\frac{T_i\{\widetilde{n}_i/\widetilde{x}_i\} \xrightarrow{\mu} T'}{D_i(\widetilde{n}_i) \xrightarrow{\mu} T'}$$

For clarity we shall not always specify all the parameters when giving parametric definitions. It is shown in (Fu and Lu 2010) that the parametric definition is equivalent to the replication in terms of expressive power.

A binary relation $\mathcal{R}$ on the set of the π^{def}-processes is *equipollent* if whenever $P\mathcal{R}Q$ then $\exists\mu \neq \tau.P \Longrightarrow\xrightarrow{\mu}$ if and only if $\exists\mu' \neq \tau.Q \Longrightarrow\xrightarrow{\mu'}$, where $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$. It is *extensional* if it is closed under composition and localization. It is a *weak bisimulation* if it satisfies the following *weak bisimulation property*:

— If $Q\mathcal{R}^{-1}P \xrightarrow{\tau} P'$ then $Q \Longrightarrow Q'\mathcal{R}^{-1}P'$ for some Q';
— If $P\mathcal{R}Q \xrightarrow{\tau} Q'$ then $P \Longrightarrow P'\mathcal{R}Q'$ for some P'.

The *weak bisimilarity* $\approx$ is the largest extensional, equipollent, weak bisimulation on the set of the π^{def}-processes. For the present calculus, $\approx$ is precisely the barbed bisimilarity of Milner and Sangiorgi (1992).

There is a variant of the π-calculus called polyadic π-calculus (Milner 1997), in which more than one names can be passed around in a single communication. To simulate the polyadic communication in the monadic π-calculus, we introduce the following abbreviations:

$$\begin{array}{rcl} a(x_1, .., x_i).T & \stackrel{\mathrm{def}}{=} & a(v).v(x_1)...v(x_i).T, \\ \overline{a}\langle n_1, .., n_i\rangle.T & \stackrel{\mathrm{def}}{=} & \overline{a}(c).\overline{c}n_1...\overline{c}n_i.T, \\ \overline{a}(b_1, .., b_i).T & \stackrel{\mathrm{def}}{=} & \overline{a}(c).\overline{c}(b_1)...\overline{c}(b_i).T. \end{array}$$

Another abbreviation used in sequel is $\prod_{i\in\{1,..,n\}} T_i$, which stands for $T_1 \,|\, \ldots \,|\, T_n$.

Before ending this section, we introduce a structural congruence over the π-processes.

Definition 2. The structural congruence $\equiv$ is the smallest congruence over the π-terms satisfying the following equalities.

1 $S \equiv T$ whenever S is α-convertible to T,
2 $T \,|\, \mathbf{0} \equiv T$, $S \,|\, T \equiv T \,|\, S$, $R \,|\, (S \,|\, T) \equiv (R \,|\, S) \,|\, T$,
3 $!T \equiv T \,|\, !T$,
4 $(a)\mathbf{0} \equiv \mathbf{0}$, $(a)(b)T \equiv (b)(a)T$,

5 $(a)(S \mid T) \equiv S \mid (a)T$, if a does not appear global in S,
6 $(a)a(x).T \equiv \mathbf{0}$, $(a)\overline{a}m.T \equiv \mathbf{0}$,
7 $(a)!a(x).T \equiv \mathbf{0}$, $(a)!\overline{a}n.T \equiv \mathbf{0}$.

In this paper the structural congruence is introduced to help define and reason about relations on processes. It is not part of the language definition and is not quite the same as the standard congruence. The following useful fact will be used without any reference.

Lemma 3. If $S' \equiv S \xrightarrow{\mu} T$ then $S' \xrightarrow{\mu} T' \equiv T$ for some T'.

We will write $P \not\rightarrow$ to mean that P cannot do any τ-action.

2.3. *Milner's Translation*

There have been a number of encodings of different variants of the λ-calculus in the π-calculus. In this subsection we use Milner's encoding of the lazy λ-calculus as an example to explain the difficulties of interpreting all the four reduction rules of the full λ-calculus. The basic idea of Milner is to explore the fact that an abstraction term $\lambda x.M$ can be seen as a process ready to interact at the name λ. Since $\lambda x.M$ is a function it is naturally encoded as a π-term in input prefix form. Now consider the application term $(\lambda x.M)N$. Clearly the encoding of $(\lambda x.M)N$ should not be able to interact at λ at this point. But structurally the encoding of $\lambda x.M$ is a π-term in input prefix form and the interface name must be different from λ. This name has to be a local name since the only interface of the λ-terms with the outside world is λ. We conclude that the encodings of the λ-terms are parameterized on names. The following is Milner's encoding:

$$
\begin{aligned}
[\![x]\!](f) &\stackrel{\text{def}}{=} \overline{x}f, \\
[\![\lambda x.M]\!](f) &\stackrel{\text{def}}{=} f(x).f(u).[\![M]\!](u), \\
[\![MN]\!](f) &\stackrel{\text{def}}{=} (cd)([\![M]\!](c) \mid \overline{c}d.\overline{c}f \mid !d(v).[\![N]\!](v)).
\end{aligned}
$$

A closed λ-term M is coded up by $[\![M]\!](f)$. The encoding of the abstraction $\lambda x.M$ is the object $f(x).f(u).[\![M]\!](u)$. The underlying structure of $[\![MN]\!](f)$ is a binary tree. The left child $[\![M]\!](c)$ is an object accessible at the local name c. The right child $!d(v).[\![N]\!](v)$ is a method with name d. Notice that an object is in prefix form whereas a method is in replication form. The part $\overline{c}d.\overline{c}f$ is the control management attached to the root. Its role is to pass the method name d to $[\![M]\!](c)$ so that the method can be called upon as many times as necessary. The control management also assigns the name f to the final object. The encoding $[\![x]\!](f)$ is simply a call that may invoke the method named x.

The encoding as it stands is sound for two of the four rules, the β-reduction rule and the structural rule. To accommodate the partial evaluation rule, the process $[\![M]\!](u)$ in $[\![\lambda x.M]\!](f)$ must not be blocked by the input prefixes. One possibility of getting rid of the blocking is to use outputs instead of inputs. Similarly the object $[\![N]\!](v)$ in $[\![MN]\!](f)$ is blocked by the prefix and the replication operator. Both must go if the eager evaluation rule is to be respected. If the replication disappears in front of $[\![N]\!](v)$, it must reappear somewhere! One way to overcome the problem could be that the control management

forces to freeze the execution of (the descendant of) $[\![N]\!](v)$ right after the contraction of the β-redex MN has been simulated. In summary, to account for the operational semantics of the full λ-calculus, Milner's encoding must be modified to something like the following one.

$$
\begin{aligned}
[\![x]\!]\langle f\rangle &\stackrel{\text{def}}{=} \overline{x}f, \\
[\![\lambda x.M]\!]\langle f\rangle &\stackrel{\text{def}}{=} (c_x d)(\overline{f}c_x.\overline{f}d \,|\, ([\![M]\!]\langle d\rangle)\{c_x/x\}), \\
[\![MN]\!]\langle f\rangle &\stackrel{\text{def}}{=} (cd)([\![M]\!]\langle c\rangle \,|\, \mathit{App} \,|\, [\![N]\!]\langle d\rangle).
\end{aligned}
$$

If we try to design *App*, we soon realize that we had to identify a local name with another (local or global) name since there are too many output prefixes and some of the names carried by the output prefixes must be identified to simulate β-reduction. This is something completely ruled out by the semantics of the π-calculus. It could be easier if we start afresh.

3. Data Structure Defined in Pi

A data structure is an organized set of data that admits a set of predefined operations. The elements of an instance of a data structure are linked in a way prescribed by the operations on the data structure. In the π-calculus a data structure is modeled by a set of processes of a certain shape. Both the elements of the data structure and the operations on the data structure are processes. The link relationship among the elements is indicated, often implicitly, by the reference relationship. In this section we explain how to define data structures in the π-calculus and introduce the structural trees that will be used in our encoding of the λ-terms.

First of all let's see how to code up lists as π-processes. A list of paired names $(\langle u_1, v_1\rangle, \ldots, \langle u_n, v_n\rangle)$ is programmed as a list of processes linked by pointers. The element $\langle u_i, v_i\rangle$ is interpreted by the process $\overline{a_i}\langle u_i, v_i, a_{i+1}\rangle$. It is accessed through a_i. The pointer a_{i+1} is the reference of the next pair. In order to indicate the end of the list, we need a special name, say $\top$ or $\bot$[†]. In Fig. 1 the list is defined together with three operations on the list. The process $\mathit{Add}(x, y, u)$ adds the pair $\langle x, y\rangle$ to the head of the list u. The process $\mathit{Remove}(x, y, u)$ deletes the pair $\langle x, y\rangle$ from u. The process $\mathit{Find}(x, u, v)$ checks if x is the first parameter of a pair in u. If it is then output the second parameter at v; otherwise it outputs $\bot$ at v.

The above example suffices to explain the general picture. Most of the data structures are special instances of the tree structure. Instead of looking at more examples of data structures coded up in the π-calculus, we shall focus in sequel on the general tree structure.

[†] In this paper $\bot$ stands for the boolean constant false as well as the name that plays the role of false. The overloading is harmless. The symbol $\top$ is used similarly.

$$
\begin{array}{rcl}
[\![()]\!]_a & \stackrel{\text{def}}{=} & \overline{a}\langle\bot,\bot,\bot\rangle \\
[\![(\langle p_0,q_0\rangle,\ldots,\langle p_n,q_n\rangle)]\!]_a & \stackrel{\text{def}}{=} & (a')(\overline{a}\langle p_0,q_0,a'\rangle \mid [\![(\langle p_1,q_1\rangle,\ldots,\langle p_n,q_n\rangle)]\!]_{a'}) \\
\hline
\mathit{Add}(x,y,u) & = & u(zz'u').(u'')(\overline{u}\langle x,y,u''\rangle \mid \overline{u''}\langle z,z',u'\rangle) \\
\mathit{Remove}(x,y,u) & = & u(zz'u').\textbf{if } u'=\bot \textbf{ then } \overline{u}\langle z,z',u'\rangle \\
 & & \qquad \textbf{else if } x=z\wedge y=z' \textbf{ then } u'(zz'u'').\overline{u}\langle z,z',u''\rangle \\
 & & \qquad\qquad \textbf{else } (\overline{u}\langle z,z',u'\rangle \mid \mathit{Remove}(x,y,u')) \\
\mathit{Find}(x,u,v) & = & u(zz'u').(\overline{u}\langle z,z',u'\rangle \mid \textbf{if } u'=\bot \textbf{ then } \overline{v}\bot \\
 & & \qquad\qquad \textbf{else if } x=z \textbf{ then } \overline{v}z' \textbf{ else } \mathit{Find}(x,u',v))
\end{array}
$$

Fig. 1. Encoding of List

3.1. *Tree Structure*

The point of introducing tree structure in the π-calculus is that an expression or a program is naturally constructed in a tree like structure. When translating a source language in the π-calculus, the tree structure readily offers something handy, which can then be modified, improved upon and refined to obtain a correct interpretation.

A binary tree defined in the π-calculus is a process of the following form:

$$\prod_{i\in\{0,1,\ldots,k\}} \overline{n_i}\langle p_i,l_i,r_i,t_i\rangle$$

where each concurrent component $\overline{n_i}\langle p_i,l_i,r_i,t_i\rangle$ represents a node. The names appeared in $\overline{n_i}\langle p_i,l_i,r_i,t_i\rangle$ suggest the following interpretation:

— n_i is the name of the present node; in other words, it is the reference of the node $\overline{n_i}\langle p_i,l_i,r_i,t_i\rangle$;

— p_i,l_i,r_i are the names of the parent, the left child and the right child respectively;

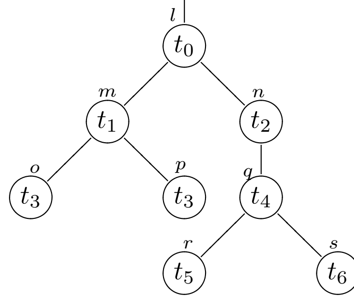
— t_i is the tag that carries additional information attached to the node.

In a particular application, there could be more than one tag to a node. Some of these tags indicate a piece of global information, others local information. A node of a tree is a term that reveals everything about itself. The revealed information includes its position in the tree, the name of an object or method attached to the node *etc.*

To indicate that a node has no parent (left child, right child), we use the special name $\bot$. Let the process T be defined as follows:

$$
\begin{array}{rcl}
T & = & \overline{l}\langle\bot,m,n,t_0\rangle \\
 & & \mid \overline{m}\langle l,o,p,t_1\rangle \mid \overline{n}\langle l,q,\bot,t_2\rangle \\
 & & \mid \overline{o}\langle m,\bot,\bot,t_3\rangle \mid \overline{p}\langle m,\bot,\bot,t_3\rangle \mid \overline{q}\langle n,r,s,t_4\rangle \\
 & & \mid \overline{r}\langle q,\bot,\bot,t_5\rangle \mid \overline{s}\langle q,\bot,\bot,t_6\rangle.
\end{array}
$$

It represents the following tree.



In this tree all the node names are exposed. In reality we are only interested in those trees where only a restricted number of the nodes are accessible. For instance $T(l)$ defined in (1) is only accessible at l.

$$T(l) = (mnopqrs)T. \tag{1}$$

It is always a good strategy that a tree is accessible only at the root.

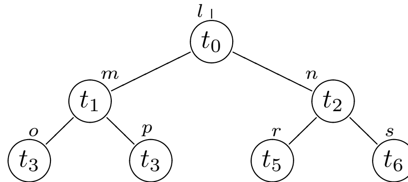
It is straightforward to implement in the π-calculus tree traversal algorithm using the depth first strategy or the breadth first one. Such a traversal is necessary for instance to check if a certain property is valid for some/all nodes of the trees. In a concurrent computation scenario, properties are dynamically established. It follows that a traversal need to be executed again and again. At the programming level this means that a while loop has to be introduced. The implementation of this idea in the π-calculus would inevitably introduce an infinite τ-loop. Most of the time this is an undesirable feature since it may preempt any useful actions like the simulation of a β-reduction.

3.2. *Operation on Tree*

The flat structure of the trees defined in the above admits easy operations on the trees. The following process

$$\begin{aligned} n(p,l,r,t).\mathbf{if}\ l \neq \bot\ \mathbf{then}\ l(p',l',r',t').(&\overline{n}\langle p,l',r',t\rangle \\ &\mid l'(p_1,l_1,r_1,t_1).\overline{l'}\langle n,l_1,r_1,t_1\rangle \\ &\mid r'(p_2,l_2,r_2,t_2).\overline{r'}\langle n,l_2,r_2,t_2\rangle) \end{aligned}$$

interacts with T and transforms it into the following tree.



This example shows that as long as we have access to one individual node of a tree we can maneuver our way through the tree to find the necessary information and use the information to manipulate the tree.

$$
\begin{array}{rcl}
Freeze(z,v) & = & z(p,l,r,t).(v_0v_1) \\
& & \textbf{begin case} \\
& & \quad l{\neq}\bot\wedge r{\neq}\bot \Rightarrow \tau.(Freeze(l,v_0)\,|\,Freeze(r,v_1) \\
& & \qquad\qquad\qquad |\,!v(p',n').(l'r')(\overline{n'}\langle p',l',r',t\rangle\,|\,\overline{v_0}\langle n',l'\rangle\,|\,\overline{v_1}\langle n',r'\rangle)) \\
& & \quad l{\neq}\bot\wedge r{=}\bot \Rightarrow \tau.(Freeze(l,v_0)\,|\,!v(p',n').(l')(\overline{n'}\langle p',l',\bot,t\rangle\,|\,\overline{v_0}\langle n',l'\rangle)) \\
& & \quad l{=}\bot\wedge r{\neq}\bot \Rightarrow \tau.(Freeze(r,v_1)\,|\,!v(p',n').(r')(\overline{n'}\langle p',\bot,r',t\rangle\,|\,\overline{v_1}\langle n',r'\rangle)) \\
& & \quad l{=}\bot\wedge r{=}\bot \Rightarrow \tau.!v(p',n').\overline{n'}\langle p',\bot,\bot,t\rangle \\
& & \textbf{end case}
\end{array}
$$

Fig. 2. Freezing-Up of Tree

When programming with trees, the nodes of a tree are attached with objects or methods. The attachment could come in two forms. The static association, implemented by the static operator composition, takes the following shape

$$\overline{n_i}\langle p_i,l_i,r_i,t_i\rangle\,|\,L(p_i,l_i,r_i,t_i).$$

Here the node $\overline{n_i}\langle p_i,l_i,r_i,t_i\rangle$ has a standby operation $L(p_i,l_i,r_i,t_i)$. The operation would not go away when the node is being visited. The static association is good enough if the methods attached to the nodes of a tree are fixed. The operation $L(p_i,l_i,r_i,t_i)$ can be designed in such a way that anybody who wants to invoke it must first get the associated node information. In this case $L(p_i,l_i,r_i,t_i)$ can only be invoked locally. The preferred use of the static associations is that they are always invoked locally. If the operations attached to the nodes of a tree are allowed to be updated, then it is more convenient to use dynamic associations, implemented by the dynamic operator choice, that take the following form

$$\overline{n_i}\langle p_i,l_i,r_i,t_i\rangle + \psi\tau.L(p_i,l_i,r_i,t_i).$$

The advantage of the dynamic association is that after the node has been visited, it could be reinstated to something like

$$\overline{n_i}\langle p_i,l_i,r_i,t_i\rangle + \psi'\tau.L'(p_i,l_i,r_i,t_i)$$

where the dynamically associated operation has been updated. It should be remarked that even if the node information is discharged once $\psi\tau.L(p_i,l_i,r_i,t_i)$ is put into operation, it is often restored later. The dynamic association can not be implemented in π^M since the choice operator is not definable in π^M (Fu and Lu 2010).

3.3. *Turning Object into Method*

In the π-calculus it is impossible to construct a general process, say Q, that produces the replicated form $!O$ of a process O by interacting with O. The reason is very simple. The process Q may contain only a finite number of names. There is no way for Q to reproduce $!a_0(x_0).\cdots.a_n(x_n).\overline{a_i}x_i$ from $a_0(x_0).\cdots.a_n(x_n).\overline{a_i}x_i$ if the set $\{a_0,\cdots,a_n\}$ is large enough. However since a data structure is a well organized set of data, it is possible

to have a π-process L that transforms every instance O of that data structure into some method from which a copy of O can be made whenever necessary, thus achieving the effect of $!O$. Two copies of O may only differ in that they are rooted at different places. Let's see an example. Suppose D is a tree defined as follows:

$$\begin{array}{rcl} D & \stackrel{\text{def}}{=} & (\bar{l}\langle\bot, m, n, t_0\rangle + t_0.Op(l, \bot, m, n, t_0)) \\ & & \mid (\overline{m}\langle l, o, p, t_1\rangle + t_1.Op(m, l, o, p, t_1)) \\ & & \mid (\overline{n}\langle l, q, \bot, t_2\rangle + t_2.Op(n, l, q, \bot, t_2)) \\ & & \mid (\overline{o}\langle m, \bot, \bot, t_3\rangle + t_3.Op(o, m, \bot, \bot, t_3)) \\ & & \mid (\overline{p}\langle m, \bot, \bot, t_3\rangle + t_3.Op(p, m, \bot, \bot, t_3)) \\ & & \mid (\overline{q}\langle n, r, s, t_4\rangle + t_4.Op(q, n, r, s, t_4)) \\ & & \mid (\overline{r}\langle q, \bot, \bot, t_5\rangle + t_5.Op(r, q, \bot, \bot, t_5)) \\ & & \mid (\overline{s}\langle q, \bot, \bot, t_6\rangle + t_6.Op(s, q, \bot, \bot, t_6)). \end{array}$$

This is a tree with dynamically associated operations. Let $D(l)$ be defined by

$$D(l) \stackrel{\text{def}}{=} (mnopqrs)D.$$

Consider the parametric definition of $Freeze(z, v)$ given in Fig. 2. The process

$$D(l) \mid Freeze(l, v)$$

may engage in a series of interactions. By the end of the engagement, the tree $D(l)$ has been turned into a blueprint of $D(l)$, which looks something like

$$\begin{array}{l} (v_0v_1)(!v(p', n').(l'r')(\overline{n'}\langle p', l', r', t_0\rangle \mid \overline{v_0}\langle n', l'\rangle \mid \overline{v_1}\langle n', r'\rangle) \\ \qquad \mid !v_0(p'', n'').(l''r'')\ldots \\ \qquad \mid !v_1(p''', n''').(l'''r''')\ldots \\ \qquad \mid \ldots). \end{array}$$

In the blueprint the operations attached to the nodes are gone. However it contains enough information so that the tree $D(l)$ can be restored as many times as necessary. If the blueprint interacts with $\overline{v}\langle\bot, l\rangle$, a replica of the tree $T(l)$, defined in (1), is produced. Then we could use the process

$$\begin{array}{rcl} Restore(l) & = & l(p', l', r', t').(Restore(l') \mid Restore(r') \\ & & \mid (\bar{l}\langle p', l', r', t'\rangle + t'.Op(l, p', l', r', t'))) \end{array}$$

to transfer $T(l)$ back to $D(l)$.

If we think of a tree structure as a piece of running program, the $Freeze$ and $Restore$ processes defined above are especially interesting. What $Freeze$ does is to terminate the execution of part of the program, place the fingerprint of its structure in store. A copy of the terminated program can be revived by $Restore$, using the fact that the operations attached to the nodes are uniform in the sense that they are parameterized over the node information. In fact the copy can be produced in such a way that it immediately replaces the subtree rooted at any specific node. The recursive instantiations of the parameters z, v

are crucial for $Freeze(z, v)$ to produce the blueprint properly. It is worth remarking that the dynamic association greatly facilitates the definition of both $Freeze$ and $Restore$.

Technically the definition of a process like $Freeze$ is tricky. Caution should be exercised to make sure that the correct instantiations of name parameters are carried out in a top-down fashion. This idea is crucial to the encoding studied in this paper.

3.4. *Operation in Tree*

Suppose an expression $op(e_1, \ldots, e_n)$ is modeled by a tree structure in the π-calculus. The root of the tree has $n + 1$ children interpreting respectively $op, e_1, \ldots, e_n$. In the general tree representation, the expression is evaluated in the following manner to admit parallel execution as much as possible:

- The node modeling op moves first. It finds its parent information using the parent link.
- Using the parent information it goes down the tree and checks if the evaluations of the expressions $e_1, \ldots, e_n$ are all finished.
- If the values of $e_1, \ldots, e_n$ are all ready, then evaluate $op(e_1, \ldots, e_n)$, otherwise either undo everything that has been done or keep on checking.

Notice that parallel executions are necessary to model the full β-reduction. The problem with the above evaluation policy is that, from the viewpoint of the interleaving semantics, the evaluation could be engaged in an infinite sequence of internal actions. This is definitely not desirable. The infinite internal chattering is caused by a strategy that goes up and down the tree to see if some statement is valid or not.

One way to prevent the unnecessary internal chattering is to introduce additional tags on the nodes. These tags not only indicate the state and the position of the nodes, they may also act as the interfaces to synchronize actions.

To simulate the operation, a tree structure should contain enough tags that enable direct interactions between the nodes. The key point is that traversal is necessary only for tree manipulation, it is neither necessary for property checking nor for potential interactions since all the useful properties are indicated by the tags on the nodes and all the potential interactions can immediately happen. The evaluations of $e_1, \ldots, e_n$ could coordinate each other such that when all are done, an interaction with the node representing op initiates the evaluation of $op(e_1, \ldots, e_n)$.

The application operation of the full λ-calculus is an example that fits into the scenario described above. In our π-interpretation of the full λ-calculus, an additional tag is used to indicate the existence of a redex.

4. Encoding in Pi

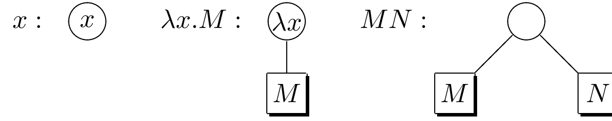
A λ-term is interpreted as a binary tree definable in the π-calculus. The nodes of the tree are associated with operations that admit a proper simulation of β-reduction. The encoding is based on the syntactical assumption that an element of $\mathcal{V}$ is an element of $\mathcal{N} \cup \mathcal{N}_v$.

$[\![L]\!]_\lambda$	$\stackrel{\text{def}}{=}$	$(n,s)([\![L]\!]^{\perp}_{n,\lambda} \mid Sem)$
$[\![x]\!]^f_{n,p}$	$=$	$L(n,p,\perp,\perp,x,f)$
$[\![\lambda x.M]\!]^f_{n,p}$	$=$	$(m,x)(L(n,p,m,m,x,f) \mid [\![M]\!]^{\perp}_{m,n})$
$[\![MN]\!]^f_{n,p}$	$=$	$(l,r)(L(n,p,l,r,\perp,f) \mid [\![M]\!]^{\top}_{l,n} \mid [\![N]\!]^{\perp}_{r,n})$

Fig. 3. Encoding of the λ-calculus

4.1. *Structural Tree for λ-Term*

The interpretation tree of a λ-term has three kinds of nodes. An application term MN is interpreted by a tree whose root has two children, the interpretations of M, N. An abstraction term $\lambda x.M$ is interpreted by a node with one child, the interpretation of M. A variable is modeled by a leaf. The following diagrams illustrate.



Formally the node information is described by the following vector:

name	*parent*	*lchild*	*rchild*	*var*	*fun*

The parameters of the vector have the following readings:

— *name* is the name of the node;
— *parent* is the name of the parent of the node; if the node does not have a parent then $parent = \perp$;
— *lchild* (*rchild*) is the name of the left (right) child of the node; if the node does not have any child then $lchild = rchild = \perp$;
— if the node corresponds to an abstraction term $\lambda x.M$ or a variable term x then $var = x$, otherwise $var = \perp$;
— if the node is the left son of its parent, then $fun = \top$, otherwise $fun = \perp$.

According to the above interpretation, the three kinds of node can be characterized as follows:

— $\perp \neq lchild \neq rchild \neq \perp$. This is an application node with two children.
— $lchild = rchild \neq \perp$. This is an abstraction node with one child.
— $lchild = rchild = \perp$. This is a variable node that is a leaf.

So the children information can tell the type of a node. But it does not say if the node is in a functional position, as the M in MN, or in a nonfunctional position, as the M in $\lambda x.M$ or in NM. This additional information is provided by the parameter fun.

4.2. *Encoding of λ-Term*

The encoding of the λ-calculus is given in Fig. 3. The interpretation $[\![L]\!]_\lambda$ of λ-term L contains a name λ which is used to access the structural tree, and is accessible only if

L is an abstraction term. The process $[\![L]\!]_\lambda$ also makes use of the global names $\bot, \top$, playing the role of the logical values. However neither $\bot$ nor $\top$ appears as an interface name in $[\![L]\!]_\lambda$.

In the standard semantics of the λ-calculus, β-reduction is contracted one at a time, which goes along with the interleaving semantics quite well. In the translation the semaphore Sem is used to prevent the simulations of two β-reductions from interfering with each other. It is given by the following recursive definition:

$$\begin{aligned} Sem &= \overline{s}.Sem^-, \\ Sem^- &= s.Sem. \end{aligned}$$

The simulation of a β-reduction begins by turning the positive state Sem to the negative state Sem^-. When the simulation ends, it turns Sem^- back to Sem. The encoding $[\![M]\!]^f_{n,p}$ defines a node of the structural tree. The subscript n, p are the name of the present node and the name of its parent respectively. The superscript f indicates if the node is the left child of its parent or not. The encoding also makes use of the global name s. The translation $[\![x]\!]^f_{n,p}$ is a leaf of the structural tree. The interpretation $[\![\lambda x.M]\!]^f_{n,p}$ is a tree, the root of which corresponds to the abstraction operator λx and has only one child; and the child is the root of the subtree $[\![M]\!]^\bot_{m,n}$. The structure of $[\![MN]\!]^f_{n,p}$ should now be clear. It is a tree with two immediate subtrees $[\![M]\!]^\top_{l,n}$, $[\![N]\!]^\bot_{r,n}$. The superscript $\top$ of $[\![M]\!]^\top_{l,n}$ indicates that M is in the function position of MN, whereas the superscript $\bot$ in $[\![N]\!]^\bot_{r,n}$ means that N is in the value position of MN.

The process $L(n, p, l, r, v, f)$, the node named n, is defined in Fig. 4. The intended meanings of the parameters of $L(n, p, l, r, v, f)$ have been explained in Section 4.1. It consists of two parts, one pinpointing the position of the node and the other defining the associated operation that can be enacted at the node. The process $\overline{n}\langle p, l, r, v, f\rangle$ declares itself and proclaims the control information related to the node. Different kinds of nodes have different influences on the tree structure.

— $l = r = \bot$. In this case $L(n, p, l, r, x, f)$ must be a leaf of the tree, standing for the variable x. The process $(a)\overline{v}\langle n, p, f, a\rangle.\overline{a}\langle\bot, \bot, \bot\rangle$ simply makes a copy of the frozen tree rooted at x. It will become clear that the frozen tree is much more complicated than the one in the previous section.

— $l = r \neq \bot \wedge f{=}\top$. This is an abstraction node that is the left child of its parent, corresponding to the left hand side of an application term. The operation of the node $Abs(n, p, l, x, f)$ can immediately kick off a β-reduction.

— $l = r \neq \bot \wedge p{=}\lambda$. This is an abstraction node that happens to be the root. In this case $RAbs(n, p, l, x, f)$ can start a β-reduction if the environment provides a term at the special interface λ. The associated operation is not the same as the one in the previous case. This should not be a problem since we are only interpreting closed λ-terms.

— $l = r \neq \bot \wedge p \neq \lambda \wedge f = \bot$. This is an abstraction node that is the right child or the only child of its parent. It cannot invoke any β-reduction. So it does not change the structure of the tree.

— $l \neq r$. This is an application node that does not invoke any action. Its role is to organize the control flow of the reduction.

$$
\begin{array}{lll}
L(n,p,l,r,v,f) & = & \overline{n}\langle p,l,r,v,f\rangle \;+\; \textbf{begin case} \\
& & \quad l = r = \bot \Rightarrow (a)\overline{v}\langle n,p,f,a\rangle.\overline{a}\langle \bot,\bot,\bot\rangle \\
& & \quad l = r \neq \bot \wedge f = \top \Rightarrow Abs(n,p,l,v,f) \\
& & \quad l = r \neq \bot \wedge p = \lambda \Rightarrow RAbs(n,p,l,v,f) \\
& & \textbf{end case}
\end{array}
$$

$$
\begin{array}{l}
Abs(n,p,m,x,f) \;=\; s.p(p^1,l^1,r^1,v^1,f^1).m(p_1,l_1,r_1,v_1,f_1).(L(p,p^1,l_1,r_1,v_1,f^1) \mid \\
\quad \textbf{begin case} \\
\quad\quad l_1 = r_1 \neq \bot \Rightarrow l_1(p_2,l_2,r_2,v_2,f_2).(L(l_1,p,l_2,r_2,v_2,f_2) \\
\quad\quad\quad \mid (b)(Backup(r^1,x,b,\bot) \mid b.\overline{s})) \\
\quad\quad l_1 \neq r_1 \Rightarrow l_1(p_2,l_2,r_2,v_2,f_2).r_1(p_2',l_2',r_2',v_2',f_2').(L(l_1,p,l_2,r_2,v_2,f_2) \\
\quad\quad\quad \mid L(r_1,p,l_2',r_2',v_2',f_2') \mid (b)(Backup(r^1,x,b,\bot) \mid b.\overline{s})) \\
\quad \textbf{end case})
\end{array}
$$

$$
\begin{array}{lll}
RAbs(n,\lambda,m,x,f) & = & \lambda(z).s.m(p_1,l_1,r_1,v_1,f_1). \\
& & \quad (b)(Backup(z,x,b,\top) \mid b.\overline{s}.L(m,\lambda,l_1,r_1,v_1,\bot))
\end{array}
$$

$$
\begin{array}{l}
Backup(n,x,b,e) \;=\; n(p,l,r,v,f). \\
\quad \textbf{begin case} \\
\quad\quad l = \bot \vee r = \bot \Rightarrow \overline{b}.!x(n',p',f',a).(o)(Find(v,a,o) \mid o(v'). \\
\quad\quad\quad \textbf{begin case} \\
\quad\quad\quad\quad v' = \bot \wedge e = \top \Rightarrow \tau.[\![\Omega]\!]^{f'}_{n'p'} \\
\quad\quad\quad\quad v' = \bot \wedge e = \bot \Rightarrow \tau.L(n',p',\bot,\bot,v,f') \\
\quad\quad\quad\quad v' \neq \bot \Rightarrow \tau.L(n',p',\bot,\bot,v',f') \\
\quad\quad\quad \textbf{end case}) \\
\quad\quad l = r \neq \bot \Rightarrow \tau.(x'b')(Backup(l,x',b',e) \mid b'.\overline{b}.!x(n',p',f',a). \\
\quad\quad\quad (m'a'v')(L(n',p',m',m',v',f') \mid \overline{x'}\langle m',n',\bot,a'\rangle \mid \overline{a'}\langle v,v',a\rangle)) \\
\quad\quad \bot \neq l \neq r \neq \bot \Rightarrow \tau.(x_0x_1b_0b_1)(Backup(l,x_0,b_0,e) \mid Backup(r,x_1,b_1,e) \\
\quad\quad\quad \mid b_0.b_1.\overline{b}.!x(n',p',f',a).(l'r')(L(n',p',l',r',\bot,f') \\
\quad\quad\quad \mid \overline{x_0}\langle l',n',\top,a\rangle \mid \overline{x_1}\langle r',n',\bot,a\rangle)) \\
\quad \textbf{end case}
\end{array}
$$

Fig. 4. Definition of Node Operation

We remark that the process $L(n, p, l, r, x, f)$ can be seen as an enriched solution for *App* mentioned in Section 2.

The $Backup(n, x, b, e)$ process makes a replica, or a blueprint, of the tree that codes up a λ-term. The parameters should be understood as follows:

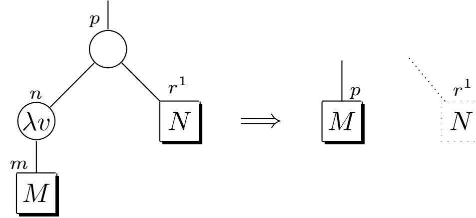
— n is the name of the root of the tree to be replicated;
— x is the name that may be used to access to the replica;
— b is the control name that releases the replica after it has been completely produced;
— e indicates if the tree rooted at n is from environment of not.

The operational behavior of $Backup(n, x, b, e)$ will be analyzed in sequel.

4.3. *Simulation of β-Reduction*

When $L(n, p, l, r, v, f)$ is an abstraction node with $f = \top$, a β-redex is present. In this case $L(n, p, l, r, v, f)$ can set the semaphore to Sem^- and begin to simulate the β-reduction. The simulation consists of two phases.

— In the first phase the child m of the node n is moved to the position of the parent p of the node n. The node n is deleted. The node name m is dropped. The right child r^1 of p is temporarily detached. The change of the tree structure is depicted as follows:



— In the second phase the subtree rooted by r^1 is backed up. This is achieved by the process $Backup(r^1, v, b, \bot)$ that replicates the subtree from the root to the leaves in such a way that a copy of the subtree can be readily generated by providing a new name of the root and its parent's name. The parameter b is to make sure that the simulation of the β-reduction is not complete until every node of the subtree is backed up. When the backup is over, Sem^- will be set back to Sem.

If $L(n, p, l, r, v, f)$ is an abstraction node situated at the root, it can backup a λ-term from the environment. This situation is indicated by instantiating e by $\top$, where e is the fourth parameter of *Backup*. In this case the backup process need to do something more. The extra work is to make sure that the result of the backup is the interpretation of a well-defined *closed* λ-term. It achieves that by treating all the interpretations of free variables as if they were the interpretations of the closed λ-term Ω. In order to do that properly, a tree structure is introduced to record the interpretations of the closed variables of the λ-term, reflecting naturally the nested structure of the closed variables. When the backup comes down to a leaf that codes up the variable v, it checks the status of that v in the term being backed up using the process *Find* defined in Fig. 1. There are three possibilities:

— If *Find* returns with $v' = \bot$ and $e = \top$, it means that the term being backed up comes from the environment and that the variable v is not the interpretation of a bound variable. In this case we substitute Ω for v in the backup.
— If *Find* returns with $v' = \bot$ and $e = \bot$, it means that the term being backed up comes from within and that v is not bound in that subterm. In this case the variable v should be backed up since every restored copy shares the same v.
— If the name v' returned by *Find* is not $\bot$, it means that v correspond to a closed variable in the λ-term being backed up. In this case a new local name must be produced every time a copy of the backup is made.

The condition $l = \bot \vee r = \bot$ in the definition of *Backup* works both for the terms from within and the terms from without. The term from the environment might be corrupted. But as long as one of l, r is $\bot$, the backup procedure will regard it as representing a variable.

The reader might wonder why the tree a of the paired names must be revisited every time a copy of the replica is restored, and why the tree is even necessary in the restore time. The point is that bound variables in different copies must not confuse. The nodes of the tree named a are dispersed underneath different replication operators so that the second parameters of the paired names are all localized underneath the replication operators.

After the simulation of $(\lambda x.M)N \longrightarrow M\{N/x\}$ the process $Backup(n, x, b, e)$ has transformed the interpretation of $(\lambda x.M)N$ into a process of the form:

$$
\begin{array}{lll}
(xx_0x_1\ldots x_k)(\ldots & | & !x(n', p', f', a).(\ldots \mid \overline{x_0}\langle\ldots\rangle \mid \ldots) \\
 & | & !x_0(n_0', p_0', f_0', a_0).(\ldots \mid \overline{x_1}\langle\ldots\rangle \mid \ldots) \\
 & | & \ldots \\
 & | & !x_k(n_k', p_k', f_k', a_k).(\ldots)).
\end{array}
$$

An occurrence of the variable x in M is interpreted as $(a)\overline{x}\langle n, p, f, a\rangle.\overline{a}\langle\bot\bot\bot\rangle$. It can make a copy of the interpretation of N by interacting with $!x(n', p', f', a)._$, starting a chain of replication. Notice that since *Find* works from leaf to root, the *Backup* process can deal with terms like $\lambda y.(\lambda y.y)y$ properly.

5. Property of Encoding

In this section, we study the behaviors of the encoding. These behaviors include what operations the translations can perform and what forms they may evolve into. This is a preamble to the next section.

By the definition of the encoding, λ is the only public interface of $[\![L]\!]_\lambda$ for every closed λ-term L. So the possible actions of $[\![L]\!]_\lambda$ are either $\xrightarrow{\tau}$ or $\xrightarrow{\lambda z}$ for some name z.

— If the next action of $[\![L]\!]_\lambda$ is a τ-action, then the simulation of $(\lambda x.M)N \to M\{N/x\}$ is initiated by this action and will be followed by a number of τ-actions carried out in two stages.
  – In *reduction stage*, the tree of $(\lambda x.M)N$ is adjusted to the tree of M, whereas the tree of N is being freezed up.

- In *replication stage*, every occurrence of x in M is replaced by a replica of N.

The reduction stage is not immediately followed by the replication stage. This is because a replication of N is fired by each occurrence of x. These replications may not only interleave among themselves, but also interleave with other reduction stages.

— If the next action of $[\![L]\!]_\lambda$ is a λz-action, then $L \equiv \lambda x.M$ for some M and the translation is ready to input a λ-term from environment. Suppose $[\![L]\!]_\lambda \xrightarrow{\lambda q} P$ for some P. By synchronizing with the semaphore, the process P can start to simulate the input of a λ-term N from the environment. This simulation may also be classified into two stages.

- In *input stage*, the tree of $\lambda x.M$ is adjusted to the tree of M, whereas the tree structure of the 'λ-term N' provided by the environment is being freezed up. The term N might contain a few grammar errors. These errors are corrected in the input stage.
- In *replication stage*, every occurrence of x in M is replaced by a replica of the corrected version of N.

We shall analyze these two cases in detail in Section 5.1 and Section 5.2 respectively. To do the analysis, we find it helpful to introduce some additional notations and definitions.

— To simplify the account, we shall pretend that we are working with the polyadic π^{def}. We shall write say $P \xrightarrow{n(p,l,r,v,f)} P'$ to mean that the names p, l, r, v, f are received in that order at n in an atomic action.

— The translation $[\![L]\!]^f_{n,p}$ of the λ-term L has an underlying tree structure. This tree takes the following form

$$(\widetilde{v})(\widetilde{n})\prod_{i\in I} L(n_i, p_i, l_i, r_i, v_i, f_i)$$

for some finite set I, where $\widetilde{v}$ are the bound variables of L, $\widetilde{n}$ are the node names of the tree excluding the root name n and the name of its parent p. For each $i \in I$, the component $L(n_i, p_i, l_i, r_i, v_i, f_i)$ is a node of the tree and n_i is the name of the node. In sequel we shall write $T^{n,p,f}_L$ for the following tree

$$\prod_{i\in I} L(n_i, p_i, l_i, r_i, v_i, f_i). \tag{2}$$

We also write $L(n', p', l', r', v', f') \in T^{n,p,f}_L$ to indicate that $L(n', p', l', r', v', f')$ is a component of $T^{n,p,f}_L$.

— We write $RT^{n,p,f}_M$ for the composition of the nodes in $T^{n,p,f}_M$ except the root located at n.

— If N is a sub-term of L, that is $L \equiv C[N]$ for some closing λ-context, then we write

$$T^{n,p,f}_L \equiv T^{n,p,f}_{C[_]} \,|\, T^{n',p',f'}_N$$

to mean that $T^{n,p,f}_{C[_]}$ is what is left after removing the nodes of $T^{n',p',f'}_N$ from $T^{n,p,f}_L$. Here n', p' are the name and the parent's name of the sub-term N.

— Suppose $T^{z,\perp,\perp}_L$ is of the form (2). The notation $T(z, L)$ stands for the following

process

$$(\widetilde{v})(\widetilde{n}) \prod_{L(n,p,l,r,v,f)\in T_L^{z,\perp\perp}} \overline{n}\langle p,l,r,v,f\rangle$$

where $\widetilde{v}$ are the bound variables of L, and $\widetilde{n}$ are the node names excluding z.

— A *rooted binary tree* T_r is a directed graph that has the following properties:

  – There is a node, called the root, that does not have any incoming edges. Every other node has precisely one incoming edge.
  – Every node has either zero, or one, or two outgoing edges. If a node has two children, the children are ordered and are called respectively left child and right child.

  A *labeled tree* LT is a rooted binary tree with each of its node labeled by a pair $\langle n,\vartheta\rangle$, where n is a name and ϑ is either a name or $\circ$. We write $\mathbf{1}_z$ for the single node labeled tree with label $\langle z,\circ\rangle$ and **LT** for the set of all labeled trees.

— Suppose LT_1 and LT_2 are two labeled trees. We write $LT_1 \Subset LT_2$ if LT_1 can be obtained from LT_2 by cutting off from some nodes all their subtrees.

— Suppose n is the label of a specific leaf of the labeled tree LT. We write $LT\cdot(n,l,r,v)$ for the labeled tree that extends LT in the following manner:

  – If $l=\perp\vee r=\perp$ then $LT\cdot(n,l,r,v)$ is obtained from LT by changing the second parameter of the label of the specific leaf to v.
  – If $l=r\neq\perp$ then $LT\cdot(n,l,r,v)$ is obtained from LT by changing the second parameter of the label of the specific leaf to v and attaching to the leaf a single child labeled $\langle l,\circ\rangle$.
  – If $\perp\neq l\neq r\neq\perp$ then $LT\cdot(n,l,r,v)$ is obtained from LT by attaching to the specific leaf a left child labeled $\langle l,\circ\rangle$ and a right child labeled $\langle r,\circ\rangle$.

— Let LT be a labeled tree. The notation $I(LT)$ denotes the multi-set of all the first parameters of the labels of the leaves of LT whose second parameter is $\circ$.

5.1. *Reduction Action*

In this subsection we consider τ-actions of the translations. Suppose O contains the β-redex $(\lambda x.M)N$. We may indicate this fact by $O\equiv C[(\lambda x.M)N]$ where $C[_]$ is a closing context for $(\lambda x.M)N$. By the definition of the encoding and structural congruence, one has that

$$[\![O]\!]_\lambda \equiv (s)(\widetilde{x})(\widetilde{n})(Sem\,|\,T_O^{n,\lambda,\perp})$$

where $\widetilde{x}$ are the bound variables in O and $\widetilde{n}$ are the internal node names of the tree $T_O^{n,\lambda,\perp}$. We can rearrange the bound names in $T_O^{n,\lambda,\perp}$ by applying structural congruence rules.

$$\begin{aligned} T_O^{n,\lambda,\perp} &\equiv T_{C[_]}^{n,\lambda,\perp}\,|\,T_{(\lambda x.M)N}^{n',p',f'}, \\ T_{(\lambda x.M)N}^{n',p',f'} &\equiv L(n',p',l,r,\perp,f')\,|\,L(l,n',m,m,x,\top)\,|\,T_M^{m,l,\perp}\,|\,T_N^{r,n',\perp}. \end{aligned}$$

The β-reduction $C[(\lambda x.M)N]\to C[M\{N/x\}]$ is simulated in two stages.

5.1.1. *Reduction Stage* A β-redex is resolved in a reduction stage. By the end of a reduction stage all the occurrences of a bound variable are tied to the replicated form of the encoding of a λ-term. This is achieved mainly by the *Backup* process. By the definition of $L(n,p,l,r,v,f)$ given in Fig. 4,

$$\begin{array}{rcl} L(l,n',m,m,x,\top) & \stackrel{s}{\longrightarrow} & L_1 \equiv n'(p^1,l^1,r^1,v^1,f^1).\cdots, \\ Sem & \stackrel{\overline{s}}{\longrightarrow} & Sem^-. \end{array}$$

So the reduction stage begins by setting Sem to the state Sem^-:

$$[\![O]\!]_\lambda \stackrel{\tau}{\longrightarrow} P_1 \quad \equiv \quad (s)(\widetilde{x})(\widetilde{n})(Sem^- \,|\, T^{n,\lambda,\perp}_{C[_]} \,|\, L(n',p',l,r,\perp,f') \,|\, L_1 \,|\, T^{m,l,\perp}_M \,|\, T^{r,n',\top}_N).$$

The process L_1 may interact with $L(n',p',l,r,\perp,f')$ and $T^{m,l,\perp}_M$. According to the structures of M, P_1 may either perform two τ-actions in the case of $M \equiv w$ for some w, or three τ-actions in the case of $M \equiv \lambda w.M_1$ for some w, M_1, or four τ-actions in the case of $M \equiv M_1M_2$ for some M_1, M_2. After these τ-actions, P_1 has evolved into P_2, i.e. we have

$$\begin{array}{rcl} P_1 \stackrel{\tau}{\Longrightarrow} P_2 & \equiv & (s)(\widetilde{x})(\widetilde{n})(Sem^- \,|\, T^{n,\lambda,\perp}_{C[_]} \,|\, T^{n',p',f'}_M \,|\, (b)(Backup(r,x,b,\perp) \,|\, b.\overline{s}) \,|\, T^{r,n',\top}_N) \\ & \equiv & (s)(\widetilde{x})(\widetilde{n})(Sem^- \,|\, T^{n,\lambda,\perp}_{C[M]} \,|\, (b)(Backup(r,x,b,\perp) \,|\, b.\overline{s}) \,|\, T^{r,n',\top}_N). \end{array}$$

Now $Backup(r,x,b,\perp)$ is able to input at r. It has three kinds of transition described as follows, where $Copy(x,x_0,x_1,w,e)$ abbreviates $!x(n',p',f',a).\cdots$.

— The first case of $Backup(n,x,b,e)$ applies. This is when $x_0 = x_1 = \perp$.

$$Backup(r,x,b,\perp) \xrightarrow{r(p,\perp,\perp,w,f)} \stackrel{\overline{b}}{\longrightarrow} \quad Copy(x,\perp,\perp,w,\perp).$$

— The second case of $Backup(n,x,b,e)$ applies. This is when $x_0 = x_1 \neq \perp$.

$$\begin{array}{rcl} Backup(r,x,b,\perp) & \xrightarrow{r(p,m,m,w,f)} \stackrel{\tau}{\longrightarrow} & (x'b')(b'.\overline{b}.Copy(x,x',x',w,\perp) \\ & & |\, Backup(m,x',b',\perp)). \end{array}$$

— The third case of $Backup(n,x,b,e)$ applies. This is when $x_0 \neq x_1$.

$$\begin{array}{rcl} Backup(r,x,b,\perp) & \xrightarrow{r(p,l',r',\perp,f)} \stackrel{\tau}{\longrightarrow} & (x_0x_1b_0b_1)(b_0.b_1.\overline{b}.Copy(x,x_0,x_1,w,\perp) \\ & & |\, Backup(l',x_0,b_0,\perp) \,|\, Backup(r',x_1,b_0,\perp)). \end{array}$$

By the above analysis, $Backup(r,x,b,\perp)$ must interact with $T^{r,n',\top}_N$, reading the nodes of the tree recursively from root to leaf. When it reaches a leaf, it backs up and unlocks all the b_i's in a backward fashion. Process P_2 evolves into the following process P_3 when the prefix b before $\overline{s}$ is demolished.

$$P_2 \stackrel{\tau}{\Longrightarrow} P_3 \equiv (s)(\widetilde{x})(\widetilde{n})(Sem^- \,|\, T^{n,\lambda,\perp}_{C[M]} \,|\, \overline{s} \,|\, [\![x := N]\!]_\perp)$$

where $[\![x{:=}N]\!]_e$ is the process

$$(x_0x_1\ldots x_k)[\![x{:=}N]\!]^{x_0x_1\ldots x_k}_e$$

and $[\![x{:=}N]\!]^{x_0x_1\ldots x_k}_e$ is the following process

$$!x(n',p',f',a)._ \,|\, !x_0(n'_0,p'_0,f'_0,a_0)._ \,|\, \ldots \,|\, !x_k(n'_k,p'_k,f'_k,a_k)._.$$

The parameter e, which equals to either $\bot$ or $\top$, is the parameter e in $Backup(n, x, b, e)$.

As the final step in the reduction stage, the process P_3 resets Sem^- to Sem in a single step:

$$P_3 \xrightarrow{\tau} P_4 \equiv (s)(\widetilde{x})(\widetilde{n'})(Sem \,|\, T^{n,\lambda,\bot}_{C[M]} \,|\, [\![x{:=}N]\!]_\bot),$$

where $\widetilde{n'}$ is obtained from $\widetilde{n}$ by removing several names, bearing in mind that in the simulation of the λ-reduction two nodes are removed and all the nodes related to N are removed.

The process $[\![x{:=}N]\!]_\bot$ is similar to the blueprint discussed in Section 3. If it interacts with $\overline{x}\langle n, p, f, a\rangle$, a replica of the tree of N is produced and is rooted at n with p being the parent node. It should be obvious that $(x)[\![x{:=}N]\!]_e \equiv \mathbf{0}$.

5.1.2. *Replication Stage* In a replication stage a copy of a replica is made in the position of an occurrence of the variable that is tied to the replica. This is achieved by providing to the replica the location name of the variable. The instantiation of an x in $C[M]$ can start as soon as $[\![x{:=}N]\!]_\bot$ is ready since a leaf node $L(n_i, p_i, \bot, \bot, x, f_i)$ in the tree of M can carry out the action $\overline{x}\langle n_i, p_i, f_i, a\rangle$ and turn into $\overline{a}\langle\bot, \bot, \bot\rangle$, where a is a newly generated local name. The process $[\![x{:=}N]\!]_\bot$ has three kinds of transitions.

— If $N \equiv N_1N_2$ for some N_1 and N_2, then

$$\begin{array}{lll} [\![x{:=}N]\!]_\bot & \xrightarrow{x(n',p',f',a)} & (\widetilde{x})([\![x{:=}N]\!]^{\widetilde{x}}_\bot \,|\, (l'r')(L(n', p', l', r', \bot, f') \\ & & \,|\, \overline{x_0}\langle l', n', \top, a\rangle \,|\, \overline{x_1}\langle r', n', \bot, a\rangle)). \end{array}$$

The application node has been copied to n'. Now the two processes $\overline{x_0}\langle l', n', \top, a\rangle$ and $\overline{x_1}\langle r', n', \bot, a\rangle$ can interact with $[\![x{:=}N]\!]_\bot$ and get the copy of N_1 and N_2 respectively.

— If $N \equiv \lambda v.N_1$ for some N_1 and v, then

$$\begin{array}{lll} [\![x{:=}N]\!]_\bot & \xrightarrow{x(n',p',f',a)} & (\widetilde{x})([\![x{:=}N]\!]^{\widetilde{x}}_\bot \,|\, (m'a'v')(L(n', p', m', m', v', f') \\ & & \,|\, \overline{x'}\langle m', n', \bot, a'\rangle \,|\, \overline{a'}\langle v, v', a\rangle)). \end{array}$$

After the abstraction node has been copied to n', the process $\overline{x'}\langle m', n', \bot, a'\rangle$ interacts with $[\![x{:=}N]\!]_\bot$ and gets the copy of N_1. The name a' is used to record the new name v' that is correlated to v. The free variable v in N_1 will be replaced by a new local name v' every time N_1 is replicated.

— If $N \equiv w$ for some w, then

$$\begin{array}{lll} [\![x{:=}N]\!]_\bot & \xrightarrow{x(n',p',f',a)} & (\widetilde{x})([\![x{:=}N]\!]^{\widetilde{x}}_\bot \,|\, (o)(Find(w, a, o) \,|\, o(v').\cdots)) \\ & \stackrel{\tau}{\Longrightarrow} & (\widetilde{x})([\![x{:=}N]\!]^{\widetilde{x}}_\bot \,|\, L(n', p', \bot, \bot, w, f')). \end{array}$$

Since we have the τ-action

$$(a)(Find(w, a, o) \,|\, \overline{a}\langle\bot, \bot, \bot\rangle) \xrightarrow{\tau} \overline{a}\langle\bot, \bot, \bot\rangle \,|\, \overline{o}\langle\bot\rangle,$$

a leaf with variable w is copied to n'.

In order not to confuse the bound names in two copies of N, the parameter a is introduced. One can easily see that all the pointers $\{a_i\}_{i\in I}$ form a tree with the root a.

For an application node, the pointer a_i is transmitted to both the left subtree and the right subtree. For an abstraction node with variable v, a new name v' is generated and the pair (v, v') is inserted with a new pointer a_j whose parent is a_i. Pointer a_j will be transmitted to the single subtree. For a variable v, $Find(v, a_i, o)$ searches the tree from a_i to the root and returns the result on channel o. If $v' = \bot$, then a leaf with variable v is copied at n'. But if $v' \neq \bot$, then instead of v, a leaf with variable v' is copied.

When two occurrences of x are copying N, the bound names will not be confused because we assign a fresh root a for every x. So every x in M is replaced by a λ-term N' which is α-convertible to N. After all the occurrences of x in M have been replaced by N, replication stage ends. Formally

$$\begin{aligned} P_4 \Longrightarrow P_5 &\equiv (s)(\widetilde{x'})(\widetilde{n'})(Sem \,|\, (x)[\![x{:=}N]\!]_\bot \,|\, T^{n,\lambda,\bot}_{C[M\{N/x\}]}) \\ &\equiv (s)(\widetilde{x'})(\widetilde{n'})(Sem \,|\, T^{n,\lambda,\bot}_{C[M\{N/x\}]}) \\ &\equiv [\![C[M\{N/x\}]]\!]_\lambda, \end{aligned}$$

where $\widetilde{x'} = \widetilde{x} \cup \widetilde{x''} \setminus \{x\}$ and $\widetilde{x''}$ are the bound variables introduced by the copies of N for all the occurrences of x in M.

Ideally the simulation of β-reductions should happen in the following fashion: Upon the completion of a reduction stage, the replication stage begins; after all the replications are complete, a new reduction stage starts, and so on. In other words the reduction $O \to O' \to O'' \to \ldots$ is simulated by

$$[\![O]\!]_\lambda \underbrace{\stackrel{\tau}{\Longrightarrow}}_{reduction} \underbrace{\Longrightarrow}_{replication} [\![O']\!]_\lambda \underbrace{\stackrel{\tau}{\Longrightarrow}}_{reduction} \underbrace{\Longrightarrow}_{replication} \cdots .$$

Realistically the encoding can start a reduction stage as long as *Sem* is ready; and it can begin a replication stage of some x as long as $[\![x{:=}N]\!]_e$ is available. If we use $\stackrel{\tau}{\longrightarrow}_1$ to denote the τ-action performed in reduction stage, and $\stackrel{\tau}{\longrightarrow}_2$ in replication stage, then a transition sequence looks like

$$[\![O]\!]_\lambda \stackrel{\tau}{\Longrightarrow}_1 \stackrel{\tau}{\Longrightarrow}_2 \stackrel{\tau}{\Longrightarrow}_1 \stackrel{\tau}{\Longrightarrow}_2 \cdots .$$

The interleaving between replication stage and reduction stage is harmless though it makes the analysis of the simulations a little nasty. The semaphore and the fact that $L(n, p, l, r, v, f) \xrightarrow{\overline{n}\langle p,l,r,v,f\rangle} \mathbf{0}$ for every node are the basic mechanism that the interleaving does not have adversary effect on the simulation.

5.2. *Input Action*

When the λ-term being encoded is an abstraction $\lambda x.M$, the translation may also perform the input action λz. According to the definition of the encoding and the structural congruence,

$$[\![\lambda x.M]\!]_\lambda \equiv (s)(Sem \,|\, (\widetilde{x})(\widetilde{n})(L(n, \lambda, m, m, x, \bot) \,|\, T^{m,n,\bot}_M)),$$

where $\widetilde{x}$ represent all the bound variables in M and $\widetilde{n}$ all the names of the tree node. By the definition of the node,

$$L(n,\lambda,m,m,x,\bot) \xrightarrow{\lambda z} L_1 \equiv s.m(p_1,l_1,r_1,v_1,f_1).(b)(Backup(z,x,b,\top) \mid b.\overline{s}.L(m,\lambda,l_1,r_1,v_1,\bot)).$$

Consequently,

$$[\![\lambda x.M]\!]_\lambda \xrightarrow{\lambda z} P_1 \equiv (s)(Sem \mid (\widetilde{x})(\widetilde{n})(L_1 \mid T_M^{m,n,\bot})).$$

At this stage the translation can still engage in a β-reduction. But if it sets the semaphore to Sem^-, the translation moves to an input stage, followed by a replication stage.

5.2.1. *Input Stage* In the input stage a replica of the encoding of a term imported from the environment is made. This is again achieved mainly by the *Backup* process. To start with, P_1 consumes the prefix s in L_1 to kick off the input stage, and then reads the root of T_M.

$$P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} Q_0 \equiv (s)(Sem^- \mid (\widetilde{x})(\widetilde{n})(b)(Backup(z,x,b,\top) \mid b.\overline{s}.L(m,\lambda,l_1,r_1,v_1,\bot) \mid RT_M^{m,n,\bot})).$$

The root m will not be reinstated before $Backup(z,x,b,\top)$ ends. This is different from the situation in the reduction stage where tree structure is adjusted at the same time the process $Backup(r,x,b,\bot)$ begins. The reason is that the root of $T_M^{m,n,\bot}$ might also be an abstraction node. If we change its parent n into λ, then a new action $\xrightarrow{\lambda z'}$ for some z' is possible. The action $\xrightarrow{\lambda z'}$ may be harmless to the correctness of the encoding. But we prefer the present encoding for its robustness.

The process $Backup(z,x,b,\top)$ works in a similar way as $Backup(z,x,b,\bot)$. It backs up the tree rooted at z from the environment. For each closed λ-term N, we have

$$(z)(Q_0 \mid T(z,N)) \stackrel{\tau}{\Longrightarrow} (s)(\widetilde{x})(\widetilde{n})(Sem \mid T_M^{n,\lambda,\bot} \mid [\![x{:=}N]\!]_\top).$$

While we are completely assured that the grammar of $T(z,N)$ is in its correct form, we are not at all certain if the term being imported from the environment is good. Non-well-definedness might come in several forms.

— The term might contain free variables; the name used to denote these free variables might even be $\bot$, $\top$ or λ.
— The tree nodes do not have the desired format.
— The node names could be name variables.
— Two nodes might share a same name.
— A node might not have any child even though neither its left child parameter nor its right child parameter is the special name $\bot$.

Against all these odds, the process $Backup(z,x,b,\top)$ works correctly by essentially *not using* any names imported from the environment in any *prefix*. We now explain how the process $Backup(z,x,b,\top)$ corrects these syntactical errors by considering all the possibilities of the input action $\xrightarrow{n(p,l,r,v,f)}$.

— Let's see the process $Backup(n, x, b, e)$ defined in Fig. 3. The names received to instantiate the parameters p, f of the prefix $n(p, l, r, v, f)$ never appear in the rest of the process. In fact the parent and the functionality information of a node is useless for $Backup(n, x, b, e)$. The information conveyed in the four parameters n, l, r, v is enough for the backup process.
— If $l = \bot \wedge r \neq \bot$ or $l \neq \bot \wedge r = \bot$, an incorrect form of a node is encountered. The process $Backup(z, x, b, \top)$ handles it as if it were a leaf.
— If $v = \bot$ or $v = \top$ or $v = \lambda$, these two names are very special in our encoding, but it does not matter if they are imported as v in an abstraction node or a leaf because in the stage of replacement, the variable in an abstraction node will be renamed and the leaf with free variables will be replaced by Ω.

Consequently $Backup(z, x, b, \top)$ acts correctly to all inputs from the environment.

Since environments do not always provide sufficient information, the backup procedure may either succeed or get stuck. In the case of a successful backup, one has the following transition:

$$\begin{array}{lcl} Q_0 & \xrightarrow{n_0(p_0,l_0,r_0,v_0,f_0)} & Q_1 \\ & \stackrel{\tau}{\Longrightarrow} & Q_1' \\ & \xrightarrow{n_1(p_1,l_1,r_1,v_1,f_1)} & Q_2 \\ & \stackrel{\tau}{\Longrightarrow} & Q_2' \\ & & \vdots \\ & \xrightarrow{n_k(p_k,l_k,r_k,v_k,f_k)} & Q_k \\ & \stackrel{\tau}{\Longrightarrow} & Q_k' \\ & \equiv & (s)(Sem^- \mid (\widetilde{x})(\widetilde{n})(\overline{s}.L(m, \lambda, l_1, r_1, v_1, \bot) \mid RT_M^{m,n,\bot}) \mid [\![x{:=}N]\!]_\top), \end{array}$$

where $n_0 = z$, $k \geq 1$ and N is a closed λ-term. After finishing with all the leaves of a tree, the process Q_k has broadcasted this fact upward through the tree, with the help of b_i's, to release $[\![x{:=}N]\!]_\top$. Finally the process Q_k' resets Sem^- to Sem by the following action

$$Q_k' \xrightarrow{\tau} P_3 \equiv (s)(Sem \mid (\widetilde{x})(\widetilde{n})(T_M^{m,\lambda,\bot} \mid [\![x{:=}N]\!]_\top)),$$

ending the input stage. So a successful backup always imports, as it were, a closed λ-term from the environment, no matter how corrupted the information provided by the environment is.

The backup procedure may fail when it is ready to read node information but the environment refuses to provide any. In this case nothing can ever happen. One might wonder why it is not a good idea to let the input stage stuck once corruption is detected. The truth is that a more complex encoding would be necessary to implement that strategy since there are many ways an input term may be corrupted.

5.2.2. *Replication Stage* A replication stage in current situation is almost the same as the replication stage in Section 5.1.2. The only difference is that the parameter e in $[\![x{:=}N]\!]_e$

takes the value $\top$ in the present case. As in the previous situation, one has

$$\begin{aligned} P_3 \Longrightarrow P_4 &\equiv (s)(\widetilde{x'})(\widetilde{n'})(Sem \,|\, (x)[\![x{:=}N]\!]_\top \,|\, T^{n,\lambda,\bot}_{M\{N/x\}}) \\ &\equiv (s)(\widetilde{x'})(\widetilde{n'})(Sem \,|\, T^{n,\lambda,\bot}_{M\{N/x\}}) \\ &\equiv [\![M\{N/x\}]\!]_\lambda, \end{aligned}$$

where $\widetilde{x'} = \widetilde{x} \cup \widetilde{y} \backslash \{x\}$, $\widetilde{y}$ contain the newly generated bound variables in every occurrence of N, and $\widetilde{n'}$ are the names of the nodes of the original tree apart from the root name plus the names of the nodes in the translation of N.

Like in the last subsection, the replication stage interleaves with other stages. Again this kind of interleaving is harmless.

5.3. *Correspondence Property*

It is not difficult to see that the key step of the simulation of a β-reduction is indicated by the change of the state of the semaphore. By setting the semaphore from the positive state Sem to the negative state Sem^-, the interpretation initiates the simulation. After the semaphore is set back to Sem, the translation signals the completion of the simulation. This suggests to classify all the descendants of the translations according to the state of the semaphore. If M is an abstraction term $\lambda x.M'$, then the interpretation of M can perform a labeled action λz for each z. After the labeled action, the descendants of the interpretation can also be classified by the two states of the semaphore. These observations are formalized in the following definition.

Definition 3. Let $M, \lambda x.M'$ be closed λ-terms, z be a name and LT be a labeled tree. The sets $\mathcal{T}_M$, $\mathcal{B}_M$, $\mathcal{LT}^z_{\lambda x.M'}$, $\mathcal{LB}^z_{\lambda x.M'}$ and $\mathcal{L}^{LT}_{\lambda x.M'}$ are defined by the following inductions.

1 $[\![M]\!]_\lambda \in \mathcal{T}_M$.

2 Suppose $P \in \mathcal{T}_M$.

(a) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (\widetilde{n})(Sem \,|\, Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, then $P' \in \mathcal{T}_M$.

(b) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (\widetilde{n})(Sem^- \,|\, Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, and if there exists some Q' such that $Q \Longrightarrow Q' \not\rightarrow$ and

$$(\widetilde{n})(Sem^- \,|\, Q') \stackrel{\tau}{\longrightarrow} (\widetilde{n})(Sem \,|\, Q'') \equiv [\![M']\!]_\lambda$$

for some M', then $P' \in \mathcal{B}_{M'}$.

(c) If $M \equiv \lambda x.M'$ and $P \stackrel{\lambda z}{\longrightarrow} P'$, then $P' \in \mathcal{LT}^z_M$.

3 Suppose $P \in \mathcal{B}_M$.

(a) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (\widetilde{n})(Sem^- \,|\, Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, then $P' \in \mathcal{B}_M$.

(b) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (\widetilde{n})(Sem \,|\, Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, then $P' \in \mathcal{T}_M$.

(c) If $M \equiv \lambda x.M'$ and $P \stackrel{\lambda z}{\longrightarrow} P'$, then $P' \in \mathcal{LB}^z_M$.

4 Suppose $P \in \mathcal{LT}^z_{\lambda x.M'}$.

(a) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (\widetilde{n})(Sem \,|\, Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, then $P' \in \mathcal{LT}^z_{\lambda x.M'}$.

(b) If $P \xrightarrow{\tau} P' \equiv (\widetilde{n})(Sem^- \mid Q)$ and if there exist some P'', P''' such that $P' \xrightarrow{\tau} P'' \xrightarrow{z(p,l,r,v,f)} P'''$, then $P' \in \mathcal{L}^{\mathbf{1}_z}_{\lambda x.M'}$.

5 Suppose $P \in \mathcal{LB}^z_{\lambda x.M'}$.

(a) If $P \xrightarrow{\tau} P' \equiv (\widetilde{n})(Sem^- \mid Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, then $P' \in \mathcal{LB}^z_{\lambda x.M'}$.

6 Suppose $P \in \mathcal{L}^{LT}_{\lambda x.M'}$.

(a) If $P \xrightarrow{\tau} P' \equiv (\widetilde{n})(Sem^- \mid Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, then $P' \in \mathcal{L}^{LT}_{\lambda x.M'}$.

(b) If $P \xrightarrow{n(p,l,r,v,f)} P'$, then $P' \in \mathcal{L}^{LT\cdot(n,l,r,v)}_{\lambda x.M'}$.

(c) If $P \xrightarrow{\tau} P' \equiv (\widetilde{n})(Sem \mid Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, and if there exist some Q' and some closed λ-term N such that $Q \Longrightarrow Q' \not\rightarrow$ and $(\widetilde{n})(Sem \mid Q') \equiv [\![N]\!]_\lambda$, then $P' \in \mathcal{T}_N$.

The π-processes in $\mathcal{T}_M$ are the translations of the λ-term M. So each closed λ-term is not just interpreted by a single π-process, but rather by a set of π-processes. The set $\mathcal{B}_M$ are the translations of the λ-term M with unfinished backup task of the last reduction. A process in $\mathcal{T}_M$ has no unfinished 'β-reduction'. This is indicated by the fact that the semaphore is in the positive state Sem. On the other hand, a process in $\mathcal{B}_M$ does have unfinished 'β-reduction', indicated by the negative state Sem^- of the semaphore. If the λ-term is an abstraction, then the π-processes in $\mathcal{T}_M$ and $\mathcal{B}_M$ can do the observable action λz, which takes the π-processes to $\mathcal{LT}^z_M$ and $\mathcal{LB}^z_M$ respectively. A π-process in $\mathcal{L}^{LT}_{\lambda x.M'}$ can repeatedly read the node information from the environment until all the leaves of the labeled tree are labeled with ⊥. While it is doing this, it expands the labeled tree LT and stays in $\mathcal{L}^{LT}_{\lambda x.M'}$. By the end of the procedure the shape of the labeled tree LT is precisely the shape of the interpretation tree of some closed λ-term N.

Definition 3 is meant to organize the statements of how the β-reductions of the closed λ-terms and the τ-actions of the interpretations of the λ-terms correspond. The next lemma explains the correspondence in one direction.

Lemma 4. Suppose $M \rightarrow M'$ for closed λ-terms M, M'. The following statements are valid.

1 If $P \in \mathcal{T}_M$ then $\exists P'.P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{T}_{M'}$.
2 If $P \in \mathcal{B}_M$ then $\exists P'.P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{T}_{M'}$.
3 If $P \in \mathcal{LT}^z_M$ then $\exists P'.P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{LT}^z_{M'}$.
4 If $P \in \mathcal{LB}^z_M$ then $\exists P'.P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{LT}^z_{M'}$.

Conversely the actions of the interpretations of a closed λ-term reflect the β-reductions of the λ-term. This is described in the following lemma.

Lemma 5. Suppose $M, \lambda x.M'$ are closed λ-terms.

1 There does not exist any infinite τ-action sequence

$$P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_i \xrightarrow{\tau} \cdots$$

in any of $\mathcal{T}_M$, $\mathcal{B}_M$, $\mathcal{LT}^z_{\lambda x.M'}$, $\mathcal{LB}^z_{\lambda x.M'}$ and $\mathcal{L}^{LT}_{\lambda x.M'}$.

2 Suppose $P \in \mathcal{T}_M$.

(a) If $P \xrightarrow{\tau} P'$ then either $P' \in \mathcal{T}_M$ or M_1 exists such that $M \to M_1$ and $P' \in \mathcal{B}_{M_1}$.

(b) If $P \xrightarrow{\lambda z} P'$ then M must be an abstraction term and $P' \in \mathcal{LT}^z_M$. If M is an abstraction term, then $\exists P'.P \Longrightarrow\xrightarrow{\lambda z} P' \in \mathcal{LT}^z_M$.

(c) If $P \xrightarrow{\mu} P'$ then either $\mu = \tau$ or $\mu = \lambda z$ for some z.

3 Suppose $P \in \mathcal{B}_M$.

(a) If $P \xrightarrow{\tau} P'$ then $P' \in \mathcal{B}_M \cup \mathcal{T}_M$.

(b) If $P \xrightarrow{\lambda z} P'$ then M must be an abstraction term and $P' \in \mathcal{LB}^z_M$. If M is an abstraction term, then $\exists P'.P \Longrightarrow\xrightarrow{\lambda z} P' \in \mathcal{LB}^z_M$.

(c) If $P \xrightarrow{\mu} P'$ then either $\mu = \tau$ or $\mu = \lambda z$ for some z.

4 Suppose $P \in \mathcal{LT}^z_{\lambda x.M'}$.

(a) If $P \xrightarrow{\tau} P'$ then either $P' \in \mathcal{LT}^z_{\lambda x.M'} \cup \mathcal{L}^{\mathbf{1}_z}_{\lambda x.M'}$ or M_1 exists such that $M' \to M_1$ and $P' \in \mathcal{LB}^z_{\lambda x.M_1}$.

(b) There exists P' such that $P \xrightarrow{\tau} P' \in \mathcal{L}^{\mathbf{1}_z}_{\lambda x.M'}$.

(c) P can and can only perform τ-actions.

5 Suppose $P \in \mathcal{LB}^z_{\lambda x.M'}$.

(a) If $P \xrightarrow{\tau} P'$ then $P' \in \mathcal{LB}^z_{\lambda x.M'} \cup \mathcal{LT}^z_{\lambda x.M'}$.

(b) P can and can only perform τ-actions.

6 Suppose $P \in \mathcal{L}^{LT}_{\lambda x.M'}$.

(a) If $P \xrightarrow{\tau} P'$ then either $P' \in \mathcal{L}^{LT}_{\lambda x.M'}$ or $P' \in \mathcal{T}_{M'\{N/x\}}$ for some closed λ-term N.

(b) If $P \xrightarrow{\tau} P' \in \mathcal{T}_{M'\{N/x\}}$ for some closed λ-term N, then for every closed λ-term $\lambda x.M''$ and every $P_1 \in \mathcal{L}^{LT}_{\lambda x.M''}$ some P_1' exists such that $P_1 \stackrel{\tau}{\Longrightarrow} P_1' \in \mathcal{T}_{M''\{N/x\}}$.

(c) If $P \xrightarrow{n(p,l,r,v,f)} P'$ then $P' \in \mathcal{L}^{LT\cdot(n,l,r,v)}_{\lambda x.M'}$. Moreover for every closed λ-term $\lambda x.M''$ and every $P_1 \in \mathcal{L}^{LT}_{\lambda x.M''}$ some P_1' exists such that $P_1 \stackrel{n(p,l,r,v,f)}{\Longrightarrow} P_1' \in \mathcal{L}^{LT\cdot(n,l,r,v)}_{\lambda x.M''}$.

(d) P can and can only do either τ-actions or input actions at names in $I(TL)$.

If we want to construct a relation $\mathcal{R}$ from Λ^0 to $\mathcal{P}$ that demonstrates the correspondence between the closed λ-terms and their translations, then inevitably $\mathcal{R}$ should contain (M, P) for every process $P \in \mathcal{T}_M \cup \mathcal{B}_M$. However, we need to throw more elements in $\mathcal{R}$ in order to close up the argument. Hence the next definition.

Definition 4. For closed λ-term $\lambda x.M$ and N, let $\mathcal{CT}^N_{\lambda x.M}$, $\mathcal{CB}^N_{\lambda x.M}$ and $\mathcal{C}^N_{\lambda x.M}$ be the smallest sets satisfying the following properties.

1 If $P \in \mathcal{LT}^z_{\lambda x.M}$, then $(z)(P \,|\, T(z,N)) \in \mathcal{CT}^N_{\lambda x.M}$.
2 If $P \in \mathcal{LB}^z_{\lambda x.M}$, then $(z)(P \,|\, T(z,N)) \in \mathcal{CB}^N_{\lambda x.M}$.
3 If $P \in \mathcal{L}^{\mathbf{1}_z}_{\lambda x.M}$, then $(z)(P \,|\, T(z,N)) \in \mathcal{C}^N_{\lambda x.M}$.
4 If $P \in \mathcal{C}^N_{\lambda x.M}$ and $P \xrightarrow{\tau} P' \equiv (\widetilde{n})(Sem^- \,|\, Q)$ for some Q and $\widetilde{n}$ with $s \in \widetilde{n}$, then $P' \in \mathcal{C}^N_{\lambda x.M}$.

By definition, the π-process $T(z,N)$ provides the static information of the interpretation structure of N. The only action it can ever do is to interact with $Backup(z,x,b,\top)$.

Intuitively an element of $\mathcal{CT}^N_{\lambda x.M}$ is an interpretation of $\lambda x.M$ that is destined to the importation of N from an environment. The difference between $\mathcal{CB}^N_{\lambda x.M}$ and $\mathcal{CT}^N_{\lambda x.M}$ is the same as that between $\mathcal{B}_M$ and $\mathcal{T}_M$. A process in $\mathcal{C}^N_{\lambda x.M}$ is an interpretation of $\lambda x.M$ that is already engaged in the importation.

The classification prescribed in Definition 4 also enjoys a kind of correspondence property.

Lemma 6. Suppose $\lambda x.M$ and N are closed λ-terms. The followings hold.

1 There does not exist any infinite τ-action sequence

$$P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_i \xrightarrow{\tau} \cdots$$

in any of $\mathcal{CT}^N_{\lambda x.M}$, $\mathcal{CB}^N_{\lambda x.M}$ and $\mathcal{C}^N_{\lambda x.M}$.

2 If $P \in \mathcal{CT}^N_{\lambda x.M} \cup \mathcal{CB}^N_{\lambda x.M} \cup \mathcal{C}^N_{\lambda x.M}$ and $M\{N/x\} \to M'$, then $P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{T}_{M'}$ for some P'.

3 Suppose $P \in \mathcal{CT}^N_{\lambda x.M}$.

(a) If $P \xrightarrow{\tau} P'$, then either $P' \in \mathcal{CT}^N_{\lambda x.M} \cup \mathcal{C}^N_{\lambda x.M}$ or M' exists such that $M \to M'$ and $P' \in \mathcal{CB}^N_{\lambda x.M'}$.

(b) There exists P' such that $P \xrightarrow{\tau} P' \in \mathcal{CT}^N_{\lambda x.M} \cup \mathcal{C}^N_{\lambda x.M}$.

(c) P can only do τ-actions.

4 Suppose $P \in \mathcal{CB}^N_{\lambda x.M}$.

(a) If $P \xrightarrow{\tau} P'$ then $P' \in \mathcal{CB}^N_{\lambda x.M} \cup \mathcal{CT}^N_{\lambda x.M}$.

(b) There exists P' such that $P \xrightarrow{\tau} P'$.

(c) P can only do τ-actions.

5 Suppose $P \in \mathcal{C}^N_{\lambda x.M}$.

(a) If $P \xrightarrow{\tau} P'$ then $P' \in \mathcal{C}^N_{\lambda x.M} \cup \mathcal{T}_{M\{N/x\}}$.

(b) There exists P' such that $P \xrightarrow{\tau} P'$.

(c) P can only do τ-actions.

In the proof of the full abstraction theorem of the next section, we need the following technical lemma.

Lemma 7. For every closed λ-term M and all P, Q in

$$\mathcal{T}_M \cup \mathcal{B}_M \cup (\bigcup_{L\{N/x\}\equiv M} \mathcal{CT}^N_{\lambda x.L} \cup \mathcal{CB}^N_{\lambda x.L} \cup \mathcal{C}^N_{\lambda x.L}),$$

one has that $P \approx Q$.

Every element of $\mathcal{T}_M \cup \mathcal{B}_M \cup (\bigcup_{L\{N/x\}\equiv M} \mathcal{CT}^N_{\lambda x.L} \cup \mathcal{CB}^N_{\lambda x.L} \cup \mathcal{C}^N_{\lambda x.L})$ can be regarded as an interpretation of the λ-term M according to our encoding. The above lemma essentially states that all the interpretations of a closed λ-term are equivalent.

The proofs of Lemma 4, Lemma 5, Lemma 6 and Lemma 7 are placed in Appendix A.

6. Correctness of Encoding

The purpose of this section is to justify the encoding defined in Section 4. It has become a consensus that such a justification should consist of two parts. Firstly the interpretation $[\![M]\!]_\lambda$ of a closed λ-term M should simulate the β-reduction of M; and it should not introduce any additional internal actions other than those simulations. This property is often referred to as the operational soundness and completeness. Secondly the encoding should relate the applicative bisimilarity on the closed λ-terms to the observational equivalence on the interpretations. This correspondence is the so-called full abstraction property. We shall discuss these two properties in the following subsections.

6.1. *Operational Soundness and Completeness*

What is required operationally for the π-process P to simulate a λ-term M? Obviously the β-reduction of M should be simulated nontrivially by the τ-actions of P. Conversely a sequence of τ-actions of P must essentially reflect the β-reduction of M. It may well be the case that P need to perform some internal adjustment before the real simulation. But these internal adjustment should end in a finite number of steps. In addition to this bisimulation property, one should have that $M \to^* \lambda x.M'$ for some M' if and only if $P \Longrightarrow\xrightarrow{\lambda z} P'$ for some fresh z and some P'. Moreover P' must also be able to simulate M' in such a way that, for each closed λ-term N, $(z)(P' \,|\, T(z,N))$ simulates $M\{N/x\}$. These remarks lead to the following definition that is meant to capture the operational soundness and completeness.

Definition 5. Let $\mathcal{R}$ be a relation from Λ^0 to $\mathcal{P}$. It is called a *subbisimilarity* if the following properties hold.

1 $\forall M \in \Lambda^0.\exists P.M\mathcal{R}P$.
2 If $P\mathcal{R}^{-1}M \to M'$ then $\exists P'.P \stackrel{\tau}{\Longrightarrow} P'\mathcal{R}^{-1}M'$.
3 If $M\mathcal{R}P \xrightarrow{\tau} P'$ then either $\exists M'.M \to M'\mathcal{R}P'$ or $M\mathcal{R}P'$.
4 If $M\mathcal{R}P_0$ and $P_0 \xrightarrow{\tau} P_1 \cdots \xrightarrow{\tau} P_i \xrightarrow{\tau} \cdots$ is an infinite sequence of τ-actions, then there must be some $k \geq 1$ and M' such that $M \to M'\mathcal{R}P_k$.
5 If $\lambda x.M\mathcal{R}P$, then $P \Longrightarrow P' \xrightarrow{\lambda z} P''$ for some fresh z and some P', P'' such that $\lambda x.M\mathcal{R}P'$ and $M\{N/x\}\ \mathcal{R}\ (z)(P'' \,|\, T(z,N))$ for every $N \in \Lambda^0$.
6 If $M\mathcal{R}P$ and $P \xrightarrow{\lambda z} P'$ for some fresh z, then $M \equiv \lambda x.M'$ for some x, M' such that $M'\{N/x\}\ \mathcal{R}\ (z)(P' \,|\, T(z,N))$ for every $N \in \Lambda^0$.

The requirements 2 and 4 imply that the encoding is termination preserving. The reader might wonder why in clauses 5 and 6 of the above definition the process $T(z,N)$ is used instead of $T_L^{z,\perp,\perp}$. The reason is that we really do not want N to reduce at this point, just like that we do not want N to reduce at the point the substitution $\{N/x\}$ is applied to M.

We are now ready to show that our encoding of the λ-calculus preserves and reflects the operational semantics.

Theorem 1. There is a subbisimilarity from Λ^0 to $\mathcal{P}$.

Proof. Let $\mathcal{R}$ be the union of the relation

$$\{(M,P) \mid M \in \Lambda^0 \wedge P \in \mathcal{T}_M \cup \mathcal{B}_M \}$$

and the relation

$$\left\{(M\{N/x\},P) \;\middle|\; \begin{array}{l} \lambda x.M, N \in \Lambda^0 \text{ and} \\ P \in \mathcal{CT}^N_{\lambda x.M} \cup \mathcal{CB}^N_{\lambda x.M} \cup \mathcal{C}^N_{\lambda x.M} \end{array} \right\}.$$

We argue that $\mathcal{R}$ is a subbisimilarity. This amounts to verifying property 1 through property 6 of Definition 5.

1 $\forall M \in \Lambda^0 . M\mathcal{R}[\![M]\!]_\lambda \in \mathcal{T}_M$.

2 Suppose $M\mathcal{R}P$ and $M \to M'$. There are two cases:
- $P \in \mathcal{T}_M \cup \mathcal{B}_M$. Then by (1) and (2) of Lemma 4 some P' exists such that $P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{T}_{M'}$.
- $P \in \mathcal{CT}^N_{\lambda x.M_1} \cup \mathcal{CB}^N_{\lambda x.M_1} \cup \mathcal{C}^N_{\lambda x.M_1}$ and $M \equiv M_1\{N/x\}$. By (2) of Lemma 6 some P' exists such that $P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{T}_{M'}$.

3 Suppose $M\mathcal{R}P \stackrel{\tau}{\longrightarrow} P'$. By making use of (2a) and (3a) of Lemma 5 and (3a), (4a) and (5a) of Lemma 6, it is easy to see that either $M\mathcal{R}P'$ or $M \to M'\mathcal{R}P'$ for some M'.

4 Suppose $M\mathcal{R}P_0$ and $P_0 \stackrel{\tau}{\longrightarrow} P_1 \cdots \stackrel{\tau}{\longrightarrow} P_i \stackrel{\tau}{\longrightarrow} \cdots$ is an infinite sequence of τ-actions. It follows from (1), (2a) and (3a) of Lemma 5, as well as (1), (3a), (4a) and (5a) of Lemma 6, that there must be some $k > 0$ and some M' such that $M \to M'\mathcal{R}P_k$.

5 Suppose $\lambda x.M\mathcal{R}P$.
- If $P \in \mathcal{T}_{\lambda x.M} \cup \mathcal{B}_{\lambda x.M}$, then it follows from (2b) and (3b) of Lemma 5 and (2c) and (3c) of Definition 3 that some P' exists such that $P \stackrel{\lambda z}{\longrightarrow} P' \in \mathcal{LT}^z_{\lambda x.M} \cup \mathcal{LB}^z_{\lambda x.M}$. Therefore $(z)(P' \,|\, T(z,N)) \in \mathcal{CT}^N_{\lambda x.M} \cup \mathcal{CB}^N_{\lambda x.M}$ by definition. Hence $M\{N/x\}\mathcal{R}(z)(P' \,|\, T(z,N))$.
- If $\lambda x.M \equiv M'\{N/y\}$ and $P \in \mathcal{CT}^N_{\lambda y.M'} \cup \mathcal{CB}^N_{\lambda y.M'} \cup \mathcal{C}^N_{\lambda y.M'}$ then by (3a), (3b), (4a), (4b), (5a) and (5b) of Lemma 6, some P' exists such that $P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{T}_{M'\{N/y\}}$. So this case is reduced to the previous case.

6 Suppose $M\mathcal{R}P$ and $P \stackrel{\lambda z}{\longrightarrow} P'$. Then P must be in $\mathcal{T}_M$ or $\mathcal{B}_M$ according to (3c), (4c) and (5c) of Lemma 6. It follows from (2b) and (3b) of Lemma 5 that $M \equiv \lambda x.M'$ for some M' and consequently $P' \in \mathcal{LT}^z_{\lambda x.M'} \cup \mathcal{LB}^z_{\lambda x.M'}$. Therefore

$$(z)(P' \,|\, T(z,N)) \in \mathcal{CT}^N_{\lambda x.M'} \cup \mathcal{CB}^N_{\lambda x.M'}$$

by definition. Hence $M'\{N/x\}\mathcal{R}(z)(P' \,|\, T(z,N))$.

We are done. □

6.2. *Full Abstraction*

Milner (1992) points out that the bisimulation equivalence on Milner's encodings of the lazy λ-calculus induces a relation weaker than β-conversion. Sangiorgi (1994; 1995) points out that the induced relation is precisely the open applicative bisimilarity on the

open λ-terms. Sangiorgi's open applicative bisimilarity extends Abramsky's applicative bisimilarity from the set of the closed λ-terms to the set of the open λ-terms. We hope to prove a similar result for the present encoding. Such a result, if attainable, would only hold for the closed λ-terms. Take for instance the open terms Ωxx and Ωx. Clearly Ωxx and Ωx are open applicative bisimilar. But $[\![\Omega xx]\!]_u$ is not bisimilar to $[\![\Omega x]\!]_u$. So we should focus on the closed λ-terms.

The difficulty in obtaining a full abstraction result is to make sure that the soundness property is valid. The most significant contribution of this paper is the soundness of our encoding with respect to the applicative bisimilarity. All the complications of our encoding are necessary in order to achieve the soundness.

Proposition 1. For all closed λ-terms M, N, $M =_a N$ implies $[\![M]\!]_\lambda \approx [\![N]\!]_\lambda$.

Proof. The relations $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ are defined as follows:

$$\mathcal{R}_1 \stackrel{\text{def}}{=} \left\{ (P,Q) \,\middle|\, \begin{array}{l} \exists M, N \in \Lambda^0.(M =_a N \wedge \\ P \in \mathcal{T}_M \cup \mathcal{B}_M \wedge Q \in \mathcal{T}_N \cup \mathcal{B}_N) \end{array} \right\},$$

$$\mathcal{R}_2 \stackrel{\text{def}}{=} \left\{ (P,Q) \,\middle|\, \begin{array}{l} \exists \lambda x.M, \lambda x.N \in \Lambda^0.\exists z.(\lambda x.M =_a \lambda x.N \wedge \\ P \in \mathcal{LT}^z_{\lambda x.M} \cup \mathcal{LB}^z_{\lambda x.M} \wedge Q \in \mathcal{LT}^z_{\lambda x.N} \cup \mathcal{LB}^z_{\lambda x.N}) \end{array} \right\},$$

$$\mathcal{R}_3 \stackrel{\text{def}}{=} \left\{ (P,Q) \,\middle|\, \begin{array}{l} \exists \lambda x.M, \lambda x.N \in \Lambda^0.(\lambda x.M =_a \lambda x.N \wedge \\ LT \text{ is a labeled tree } \wedge P \in \mathcal{L}^{LT}_{\lambda x.M} \wedge Q \in \mathcal{L}^{LT}_{\lambda x.N}) \end{array} \right\}.$$

These relation are clearly symmetric. Let $\mathcal{R}$ be defined by the following relation:

$$\mathcal{R} \stackrel{\text{def}}{=} \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3.$$

We shall prove that $\mathcal{R}$ is a weak bisimulation. By definition there are three cases.

Case I: $(P,Q) \in \mathcal{R}_1$.

— $P \in \mathcal{T}_M$. By (2c) of Lemma 5, there are two cases.

- If $P \xrightarrow{\tau} P'$, then by (2a) of Lemma 5, either $P' \in \mathcal{T}_M$ or there exists some M' such that $M \to M'$ and $P' \in \mathcal{B}_{M'}$. It follows from $M' =_a M =_a N$ that $P'\mathcal{R}_1 Q$.
- If $P \xrightarrow{\lambda z} P'$ then, by (2b) of Lemma 5 and (2c) of Definition 3, $M \equiv \lambda x.M'$ for some M' such that $P' \in \mathcal{LT}^z_M$. Since $M =_a N$, it must be the case that $N \to^* \lambda x.N'$ for some N' such that $M =_a \lambda x.N'$. Then by (1) and (2) of Lemma 4 some Q'' exists such that $Q \stackrel{\tau}{\Longrightarrow} Q'' \in \mathcal{T}_{\lambda x.N'}$. By (2b) of Lemma 5 and (2c) of Definition 3, one has that $Q'' \xrightarrow{\lambda z} Q' \in \mathcal{LT}^z_{\lambda x.N'}$. Therefore $P'\mathcal{R}_2 Q'$.

— $P \in \mathcal{B}_M$. By (3c) of Lemma 5, there are two cases.

- If $P \xrightarrow{\tau} P'$, then $P' \in \mathcal{B}_M \cup \mathcal{T}_M$ by (3a) of Lemma 5. Hence $P'\mathcal{R}_1 Q$.
- If $P \xrightarrow{\lambda z} P'$, then $M \equiv \lambda x.M'$ for some x, M' and $P' \in \mathcal{LB}^z_M$ by (3b) of Lemma 5 and (3c) of Definition 3. Since $M =_a N$, it must be the case that $N \to^* \lambda x.N'$ for some N' such that $M =_a \lambda x.N'$. Then by (1) and (2) of Lemma 4 some Q'' exists such that $Q \stackrel{\tau}{\Longrightarrow} Q'' \in \mathcal{T}_{\lambda x.N'}$. By (2b) of Lemma 5 and (2c) of Definition 3, one has that $Q'' \xrightarrow{\lambda z} Q' \in \mathcal{LT}^z_{\lambda x.N'}$. So $P'\mathcal{R}_2 Q'$.

Case II: $(P,Q) \in \mathcal{R}_2$.

— $P \in \mathcal{LT}^z_{\lambda x.M}$. By (4c) of Lemma 5, there is one case.

- If $P \stackrel{\tau}{\longrightarrow} P'$, then by (4a) of Lemma 5, either $P' \in \mathcal{LT}^z_{\lambda x.M} \cup \mathcal{L}^{\mathbf{1}_z}_{\lambda x.M}$ or there exists some M' such that $M \to M'$ and $P' \in \mathcal{LB}^z_{\lambda x.M'}$. If $P' \in \mathcal{LT}^z_M$ then $P'\mathcal{R}_2 Q$. If $P' \in \mathcal{L}^{\mathbf{1}_z}_{\lambda x.M}$, then by (1), (4a), (4b) and (5a) of Lemma 5, there exist some Q'', Q' such that $Q \Longrightarrow Q'' \in \mathcal{LT}^z_{\lambda x.N}$ and $Q'' \stackrel{\tau}{\longrightarrow} Q' \in \mathcal{L}^{\mathbf{1}_z}_{\lambda x.N}$. So $P'\mathcal{R}_3 Q'$. If $P' \in \mathcal{LB}^z_{M'}$ then $P'\mathcal{R}_2 Q$ because $M' =_a M =_a N$.

— $P \in \mathcal{LB}^z_{\lambda x.M}$. By (5b) of Lemma 5, there is one case.

- If $P \stackrel{\tau}{\longrightarrow} P'$ then $P' \in \mathcal{LT}^z_{\lambda x.M} \cup \mathcal{LB}^z_{\lambda x.M}$ by (5a) of Lemma 5. Hence $P'\mathcal{R}_2 Q$.

Case III: $(P,Q) \in \mathcal{R}_3$.

— $P \in \mathcal{L}^{LT}_{\lambda x.M}$. By (6d) of Lemma 5, there are two cases.

- According to (6a) of Lemma 5, if $P \stackrel{\tau}{\longrightarrow} P'$ then either $P' \in \mathcal{L}^{LT}_{\lambda x.M}$ or $P' \in \mathcal{T}_{M\{L/x\}}$ for some closed λ-term L. In the former case $P'\mathcal{R}_3 Q$. In the latter case $Q \stackrel{\tau}{\Longrightarrow} Q' \in \mathcal{T}_{N\{L/x\}}$ for some Q' due to (6b) of Lemma 5.
- If $P \xrightarrow{n(p,l,r,v,f)} P'$, then $P' \in \mathcal{L}^{LT\cdot(n,l,r,v)}_{\lambda x.M}$ by (6c) of Lemma 5. For the same reason, $Q \stackrel{n(p,l,r,v,f)}{\Longrightarrow} Q' \in \mathcal{L}^{LT\cdot(n,l,r,v)}_{\lambda x.N}$ for some Q'. Clearly $P'\mathcal{R}_3 Q'$.

We may now conclude that $\mathcal{R}$ is a weak bisimulation. Hence $\mathcal{R} \subseteq \approx$. It follows then from the definition of $\mathcal{R}$ that $M =_a N$ implies $[\![M]\!]_\lambda \approx [\![N]\!]_\lambda$. □

The proof of the above proposition is actually given for the polyadic π^{def}. Actually the result is also valid for the monadic π^{def}. The translation of the polyadic π^{def} to the monadic π^{def} given in Section 2.2 is not sound with respect to $\approx$. For instance $a(x,y) \,|\, b(u,v)$ is equivalent to $a(x,y).b(u,v) + b(u,v).a(x,y)$ in the polyadic π^{def}. But their translations in the monadic π^{def} are not equivalent. However if we focus on the set of the interpretations of the closed λ-terms, the encoding of Section 2.2 is fully abstract. We now explain why. If we take the set of the translations of the polyadic π^{def}-processes as the set of the observers for the monadic π^{def}-processes, then we get an equivalence relation $\approx_m$ on the set of the monadic π^{def}-processes. This equivalence is strictly weaker than $\approx$, that is $\approx \subsetneq \approx_m$. But if two bisimilar polyadic π^{def}-processes are interpretations of closed λ-terms, then their translations into the monadic π^{def}-calculus are equivalent with respect to $\approx_m$. A formal proof is tedious, although the idea is very simple. The main points are as follows: Suppose $P\mathcal{R}Q$, where $\mathcal{R}$ is the relation defined in the above proof. The set of the actions P, Q can perform may be classified into three groups.

— The first group consists of the actions that essentially simulate the β-reductions. These interactions are all carried out at the local names. The set of the observers can never interfere with these interactions. Formally one has for example

$$(a)C[a(x,y).T \,|\, \overline{a}\langle b,c\rangle.O] \approx (a)C[a(z).z(x).z(y).T \,|\, \overline{a}(e).\overline{e}(b).\overline{e}(c).O],$$

where $C[_]$ represents some environment.

— The second group has the λ-action like λz. The name λ carries only one parameter. It is automatically a monadic prefix. An observation at λ in the polyadic π^{def} is the same as in the monadic π^{def}.

— The third group contains the input actions that import the node information from the environment. Now suppose $P \in \mathcal{L}^{LT}_{\lambda x.M}$, $Q \in \mathcal{L}^{LT}_{\lambda x.N}$ and $\lambda x.M =_a \lambda x.N$. After being translated to the monadic π^{def}, P may perform an unintended corrupt actions due to the decomposition of polyadic actions to monadic actions. But the point is that Q can do precisely the same unintended corrupt actions. This is because the input actions are all contributed by the *Backup* subroutine. According to the definition of *Backup*, the effect of inputting bad information is local. Moreover *Backup* is designed in such a way that no matter what information it gets from the environment, it always discards the information it receives and produces good replicas.

In fact we do not need to refer to the encoding of the polyadic π^{def} into the monadic π^{def}. If we understand the polyadic notation as an abbreviation in the monadic calculus, the above proof of Proposition 1 can be rephrased to produce a longer proof for the encoding in the monadic π^{def}. The present proof can be seen as a shorthand version of that longer proof.

In the other direction the encoding also reflects the applicative bisimilarity in the sense of the next proposition.

Proposition 2. If $M, N \in \Lambda^0$ then $[\![M]\!]_\lambda \approx [\![N]\!]_\lambda$ implies $M =_a N$.

Proof. We prove that the symmetric relation

$$\mathcal{R} \stackrel{\mathrm{def}}{=} \{(M,N) \mid [\![M]\!]_\lambda \approx [\![N]\!]_\lambda\}$$

is an applicative bisimulation. If $M \to M'$ then $M' =_a M$ by Lemma 1. It follows from Proposition 1 that $[\![M']\!]_\lambda \approx [\![M]\!]_\lambda \approx [\![N]\!]_\lambda$ and consequently $M'\mathcal{R}N$. Therefore we only need to consider the case that at least one of M, N is an abstraction term.

Now suppose $[\![M]\!]_\lambda \approx [\![N]\!]_\lambda$ and $M \equiv \lambda x.M'$. Then $[\![M]\!]_\lambda \xrightarrow{\lambda z} P' \in \mathcal{LT}^z_{\lambda x.M'}$ for some P' and some fresh name z. This action must be simulated by

$$[\![N]\!]_\lambda \Longrightarrow Q''' \xrightarrow{\lambda z} Q'' \Longrightarrow Q' \approx P'$$

for some Q''', Q'', Q'. According to (2a), (2b), (3a) and (3b) of Lemma 5, one has that $Q''' \in \mathcal{T}_{\lambda x.N_1} \cup \mathcal{B}_{\lambda x.N_1}$ for some N_1 such that $N \to^* \lambda x.N_1$. Hence $Q'' \in \mathcal{LT}^z_{\lambda x.N_1} \cup \mathcal{LB}^z_{\lambda x.N_1}$. Using (4a) and (5a) of Lemma 5, one gets that

$$Q' \in \mathcal{LT}^z_{\lambda x.N_2} \cup \mathcal{LB}^z_{\lambda x.N_2} \cup \mathcal{L}^{\mathbf{1}_z}_{\lambda x.N_2}$$

for some N_2 such that $N_1 \to^* N_2$. By the congruence property of $\approx$, we have that

$$(z)(P' \mid T(z,L)) \approx (z)(Q' \mid T(z,L))$$

for each closed λ-term L. By definition,

$$(z)(P' \mid T(z,L)) \in \mathcal{CT}^L_{\lambda x.M'}$$

and

$$(z)(Q' \mid T(z,L)) \in \mathcal{CT}^L_{\lambda x.N_2} \cup \mathcal{CB}^L_{\lambda x.N_2} \cup \mathcal{C}^L_{\lambda x.N_2}.$$

Now

$$(z)(P' \mid T(z,L)) \approx [\![M'\{L/x\}]\!]_\lambda \tag{3}$$

and

$$(z)(Q' \mid T(z,L)) \approx [\![N_2\{L/x\}]\!]_\lambda. \tag{4}$$

The equivalence (3) can be proved by induction on the structures of P' and $T(z,L)$. Notice that the structure of P' is derived from that of M'. The equivalence (4) can be proved in a similar fashion. It follows from (3), (4) and Lemma 7 that

$$[\![M'\{L/x\}]\!]_\lambda \approx [\![N_2\{L/x\}]\!]_\lambda.$$

In summary $N \to^* \lambda x.N_2$ and $N_2\{L/x\}\mathcal{R}M'\{L/x\}$ for all $L \in \Lambda^0$.

We conclude that $\mathcal{R}$ is an applicative bisimulation. □

The full abstraction now follows from Proposition 1 and Proposition 2.

Theorem 2. Suppose $M, N \in \Lambda^0$. Then $[\![M]\!]_\lambda \approx [\![N]\!]_\lambda$ if and only if $M =_a N$.

7. Conclusion

The paper advocates a more disciplined methodology for programming with the π-calculus. The idea that the π-calculus supports naturally an object oriented paradigm has been populated by the work of Walker (1991;1995). An object is a general π-process that may or may not terminate, whereas a method is a replicated form of process that can be invoked in a potentially infinite number of times. One contribution of this paper is to formalize, in an on-the-fly manner, the procedure of turning an object into a method, the latter is supposed to be a replicated form of the object. From the technical viewpoint, the paper exploits the use of data structures to facilitate the object-to-method transfer. An object must be designed with an underlying data structure so that it can be suspended, blueprinted and restored in a replication form. The use of data structures is crucial to all the three phases. To demonstrate the power of the methodology and the technicality that comes with it, they are applied to resolve the issue of interpreting the full operational semantics of the λ-calculus.

Our encoding is given at a lower level than Milner's encoding. Two questions arise. Is this level of detail necessary? What benefit do we get from this low level interpretation? We are not in a position to give a definite answer to the first question. But as Milner (1992) points out, the structural semantics of the full λ-calculus is incompatible to that of the π-calculus. A simple structural interpretation of the former in the latter is highly unlikely. The low level programming appears inevitable. The answer to the second question is somehow related. The encoding is more an implementation than a structural translation. One may think of the π-calculus as providing a machine language upon which the 'higher order programming language', the full λ-calculus, is implemented. The significance of the present work, in the light of language implementation, is that it points out the problems and possible solutions when implementing something in the π-calculus. The extra technicality used in the encoding to achieve the full abstraction is precisely what is necessary when implementing a typed higher order language in the untyped machine language of the π-calculus. At a more fundamental level the technical novelty exhibited in the encoding is extremely useful in establishing expressiveness results. See (Fu and Zhu 2010; Fu 2010b) for interesting examples and (Fu 2010a) for a systematic exposure.

There could be many variations of the encoding. In one extreme, one could give a more deterministic encoding than the one proposed in this paper. In the deterministic variant, the backup procedure is immediately followed by an instantiation procedure whose purpose is to replace all the occurrences of the variable by the imported term. The instantiation procedure is based on a tree traversal algorithm that inspects all the nodes of the structural tree. The deterministic encoding enjoys an easier correctness proof since it is easier to construct the subbisimilarity. The proof of Theorem 1 is simplified because a lot of parallelism is removed in the deterministic encoding. But it just does not look nice. In the opposite extreme, one could imagine an encoding that does not use the semaphore at all. Such an encoding can probably be regarded as beautiful. But it would make the correctness proof a headache due to the presence of a multitude of interleaving activities. The encoding of the paper is a tradeoff between the simplicity of the encoding and the tractability of the correctness proof.

An important question remains unanswered in this paper. Can the encoding be defined in π^M? The π^{def}-calculus has two constructs that are not present in π^M, the parametric definition and the case construction. The latter can be decomposed into mismatch, match and guarded choice constructs. Let's take a look at the possibility of doing away with these constructs.

— It is a kind of a folklore that the parametric definition is equivalent to the replication in the variants of the π-calculus with only guarded choice. This issue is discussed in (Milner 1997) and is systematically examined in (Fu and Lu 2010). Parametric definition offers a more concise way of expressing a programming idea than the replication. But the encoding of this paper can be equivalently given in the variant $\pi^!$ of π^{def}, which uses the replication instead of the parametric definition.

— To model internal β-reductions, the case processes are not necessary. For instance we may add four more parameters to the definition of node. To read from a node named n, we can apply a process of the form

$$n(p, l, r, v, f, c_0, c_1, c_2, c).(c_0.C_0 \mid c_1.C_1, \mid c_2.C_2 \mid \overline{c}).$$

The names received for the parameters c_0, c_1, c_2, c are local names. If $\perp \neq l \neq r \neq \perp$, then the received names for c, c_2 are equal. In this case only C_2 will be fired. The other two cases can be similarly treated. The definition of $L(p, l, r, v, f)$ makes use of the static association rather than the dynamic association. So this part of encoding can be done in π^M.

— The real difficulty is concerned with the part of the encoding that deals with the importation of terms from the environments. The names read from the environment could be free, which renders the above strategy useless. One solution could be to change from the information retrieval viewpoint to an information verification viewpoint. An interpretation of the abstraction term $\lambda x.M$ does not import any term tree from an environment. Instead it builds up a replicated form of a term tree by making enquiry for the shape of the term tree. For instance the interpretation of $\lambda x.M$ may contain a component

$$\overline{\lambda}(u).\overline{u}(c_0).\overline{u}(c_1).\overline{u}(c_2).u(c).(c_0._ \mid c_1._ \mid c_2._ \mid \overline{c}._).$$

The process places the enquiry (c_0, c_1, c_2) on the environment. The environment answers the enquiry by sending back one of the local names c_0, c_1, c_2 as a verification code. The process then generates part of the tree according to the code and then make further enquiries if necessary. The description of this alternative approach is oversimplified, but it does outline a picture. If this idea sounds so nice, why don't we go about it? Well it has problems. For one thing, when a leaf is reached, we really need to know which variable it represents. If we cannot come up with some nice answers, we are back to square one.

We tend to believe that there does not exist a fully abstract encoding of the full λ-calculus in π^M. Our intuition is that in the absence of the match and mismatch operators there is no way to produce an encoding that is robust enough to withstand all the attacks from the environment. Proving such a negative conjecture is a much harder challenge. The expressiveness of π^M is one of the most important issues in process theory. See (Fu 2010a) for more discussions on the issue.

It is tempting to think that better encodings of the full λ-calculus can be produced using more 'advanced' variants of the π-calculus. For example the target model can be one of the typed π-calculi (Sangiorgi and Walker 2001). But if we think of it, types can only refuse syntactically wrong data, it can never rule out all the tree structures that are type safe but are logically wrong. So even in the presence of a strong type system, additional programming is still necessary to achieve the full abstraction property. One may also try to encode the full λ-calculus into the higher order π-calculus (Sangiorgi 1993b). After all the higher order communication is closer to the β-reduction than the first order communication. It is however unlikely that exercise of this kind would pay off. The higher order prefix is a sequential operator, whereas the lambda binding is not. The best one can do with the higher order π-calculus is to mimic the (first order) π-calculus when coding up the low level activities. So a kind of double encoding is present. The point we are making is that the encoding of this paper brings out the intrinsic difficulties of modeling the β-reduction in a process algebraic framework. Additional language features may well complicate rather than simplify the picture.

Acknowledgement

The authors are supported by the National Science Foundation of China (60873034, 61033002, 61003013). They would like to thank the members of BASICS for their interest in this work. The encoding described in this paper was announced in BASICS 2009. The authors would like to thank the participants of BASICS 2009 for their questions and comments. They are also grateful to the two anonymous referees for their detailed comments on the previous version of the paper.

References

Abadi, M. and Gordon, A. (1999) A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70.

Abramsky, S. (1990) The Lazy Lambda Calculus. *Research Topics in Functional Programming*, pages 65–116.

Amadio, R. and Prasad, S. (2000) Modelling IP Mobility. *Formal Methods in System Design*, 17(1):61–99.

Baldamus, M., Parrow, J. and Victor, B. (2004) Spi Calculus Translated to π-calculus Preserving May-Tests. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS'04)*, IEEE Computer Society, pages 22–31.

Barendregt, H. (1984) *The Lambda Calculus: its Syntax and Semantics.* North Holland.

Boudol, G. (1992) Asynchrony and the π-calculus. Technical Report 1702, INRIA Sophia-Antipolis.

Fu, Y. (1997) A Proof Theoretical Approach to Communication. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *Lecture Notes in Computer Science*, pages 325–335. Springer.

Fu, Y. (1999) Variations on Mobile Processes. *Theoretical Computer Science*, 221:327–368.

Fu, Y. (2010a) Theory of Interaction. Working Paper.

Fu, Y. (2010b) The Value-Passing Calculus. Working Paper.

Fu, Y and Lu, H. (2010) On the Expressiveness of Interaction. *Theoretical Computer Science*, 411:1387–1451.

Fu, Y. and Zhu, H. (2010) The Name-Passing Calculus. Working Paper.

Honda, K. and Tokoro, M. (1991) An Object Calculus for Asynchronous Communications. In *Proceedings of ECOOP'91*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer.

Honda, K. and Tokoro, M. (1991) On Asynchronous Communication Semantics. In *Proceedings of Workshop on Object-Based Concurrent Computing*, volume 615 of *Lecture Notes in Computer Science*, pages 21–51. Springer.

Merro, M. and Sangiorgi, D. (2004) On Asynchrony in Name-Passing Calculi. *Mathematical Structures in Computer Science*, 14(05):715–767.

Milner, R. (1992) Functions as Processes. *Mathematical Structures in Computer Science*, 2:119–146.

Milner, R. (1997) The Polyadic π-calculus: a Tutorial. *Theoretical Computer Science*, 198:239–249.

Milner, R., Parrow, J. and Walker, D. (1992) A Calculus of Mobile Processes, Part I and Part II. *Information and Computation*, 100(1):1–77.

Milner, R. and Sangiorgi, D. (1992) Barbed Bisimulation. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, volume 92 of *Lecture Notes in Computer Science*, pages 685–695. Springer.

Nestmann, U. and Pierce, B. (2000) Decoding Choice Encodings. *Information and Computation*, 163(1):1–59.

Palamidessi, C. (2003) Comparing the Expressive Power of the Synchronous and Asynchronous π-calculi. *Mathematical Structures in Computer Science*, 13(05):685–719.

Parrow, J. and Victor, B. (1997) The Update Calculus. *Lecure Notes in Computer Science*, pages 409–423. Springer.

Parrow, J. and Sangiorgi, D. (1995) Algebraic Theories for Name-Passing Calculi. In *Information and Computation*, 120:174–197.

Plotkin, G. (1975) Call-by-Name, Call-by-Value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159.

Sangiorgi, D. (1993a) *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, Department of Computer Science.

Sangiorgi, D. (1993b) From π-calculus to Higher-Order π-calculus – and Back. In *Proceedings of TAPSOFT'93*, pages 151–166. Springer-Verlag London, UK.

Sangiorgi, D. (1994) The Lazy λ-calculus in a Concurrency Scenario. *Information and Computation*, 111:120–153.

Sangiorgi, D. (1995) Lazy Functions and Mobile Processes. Technical Report 2515, INRIA Sophia-Antipolis, 1995.

Sangiorgi, D. (1996) π-calculus, Internal Mobility, and Agent-Passing Clculi. *Theoretical Computer Science*, 167(1-2):235–274.

Sangiorgi, D. and Walker, D. (2001) *The π Calculus: A Theory of Mobile Processes.* Cambridge University Press.

Thomsen, B. (1993) Plain CHOCS – A Second Generation Calculus for Higher Order Processes. *Acta Informatica*, 30(1):1–59.

Thomsen, B. (1995) A Theory of Higher Order Communicating Systems. *Information and Computation*, 116(1):38–57.

Walker, D. (1991) π-calculus Semantics for Object-Oriented Programming Languages. In *Proceedings of TACS'91*, volume 91 of *Lecture Notes in Computer Science*, pages 532–547. Springer.

Walker, D. (1995) Objects in the π-calculus. *Information and Computation*, 116(2):253–271.

Appendix A. Proofs of Section 5.3

The encoding given in Fig. 4 is small as a piece of program. It is not small when we want to prove some serious properties about the encoding. The interpretation of a λ-term may engage in complex interleaving activities. The important thing to notice is that the main cause of interleaving is the instantiation of variables by terms. These interleaving activities are conducted in a completely parallel fashion. The proofs given in this appendix are among those proofs in process theory that they must be delivered at an appropriate level of detail. Otherwise the size of the proofs would just be unnecessarily large. In the following proofs, a number of statements will be made without proofs. However the validity of these statements can be routinely checked at the expanse of much more space.

A.1. *Proofs of Lemma 4 and Lemma 5*

We prove Lemma 8 stated below. It can be easily seen that, apart from clauses (2d) and (4d), Lemma 8 is precisely Lemma 5 and that Lemma 4 is a corollary of clauses (1), (2d), (3a), (4d) and (5a) of Lemma 8.

In the following proofs, we will make use of some abbreviations defined in Section 5. The reader is referred to Section 5 for the definitions.

Lemma 8. Suppose $M, \lambda x.M'$ are closed λ-terms.

1 There does not exist any infinite τ-action sequence

$$P_0 \stackrel{\tau}{\longrightarrow} P_1 \stackrel{\tau}{\longrightarrow} \cdots \stackrel{\tau}{\longrightarrow} P_i \stackrel{\tau}{\longrightarrow} \cdots$$

in any of $\mathcal{T}_M$, $\mathcal{B}_M$, $\mathcal{LT}^u_{\lambda x.M'}$, $\mathcal{LB}^u_{\lambda x.M'}$ and $\mathcal{L}^{LT}_{\lambda x.M'}$.

2 Suppose $P \in \mathcal{T}_M$.

(a) If $P \stackrel{\tau}{\longrightarrow} P'$, then either $P' \in \mathcal{T}_M$ or $\exists M_1.M \to M_1 \wedge P' \in \mathcal{B}_{M_1}$.

(b) If $P \stackrel{\lambda u}{\longrightarrow} P'$, then M is an abstraction term and $P' \in \mathcal{LT}^u_M$; if M is an abstraction term, then $\exists P'.P \Longrightarrow \stackrel{\lambda u}{\longrightarrow} P' \in \mathcal{LT}^u_M$.

(c) If $P \stackrel{\mu}{\longrightarrow} P'$, then $\mu = \tau \vee \exists u.\mu = \lambda z$.

(d) If $M \to M_1$, then $\exists P'.P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{B}_{M_1}$.

3 Suppose $P \in \mathcal{B}_M$.

(a) If $P \stackrel{\tau}{\longrightarrow} P'$, then $P' \in \mathcal{B}_M \cup \mathcal{T}_M$.

(b) If $P \stackrel{\lambda u}{\longrightarrow} P'$, then M is an abstraction term and $P' \in \mathcal{LB}^u_M$; if M is an abstraction term, then $\exists P'.P \Longrightarrow \stackrel{\lambda u}{\longrightarrow} P' \in \mathcal{LB}^u_M$.

(c) If $P \stackrel{\mu}{\longrightarrow} P'$, then $\mu = \tau \vee \exists u.\mu = \lambda u$.

4 Suppose $P \in \mathcal{LT}^u_{\lambda x.M'}$.

(a) If $P \stackrel{\tau}{\longrightarrow} P'$, then either $P' \in \mathcal{LT}^u_{\lambda x.M'} \cup \mathcal{L}^{\mathbf{1}_u}_{\lambda x.M'}$ or $\exists M_1.M' \to M_1 \wedge P' \in \mathcal{LB}^u_{\lambda x.M_1}$.

(b) $\exists P'.P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{LT}^u_{\lambda x.M'} \cup \mathcal{L}^{\mathbf{1}_u}_{\lambda x.M'}$.

(c) P can and can only perform τ-actions.

(d) If $\lambda x.M' \to \lambda x.M'_1$, then $\exists P'.P \stackrel{\tau}{\Longrightarrow} P' \in \mathcal{LB}_{M'_1}$.

5 Suppose P in $\mathcal{LB}^u_{\lambda x.M'}$.

(a) If $P \stackrel{\tau}{\longrightarrow} P'$, then $P' \in \mathcal{LB}^u_{\lambda x.M'} \cup \mathcal{LT}^u_{\lambda x.M'}$.

(b) P can and can only perform τ-actions.

6 Suppose $P \in \mathcal{L}^{LT}_{\lambda x.M'}$.

(a) If $P \stackrel{\tau}{\longrightarrow} P'$, then either $P' \in \mathcal{L}^{LT}_{\lambda x.M'}$ or $P' \in \mathcal{T}_{M'\{N/x\}}$ for some closed λ-term N.

(b) If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{T}_{M'\{N/x\}}$ for some closed λ-term N, then for every closed λ-term $\lambda x.M''$ and every $P_1 \in \mathcal{L}^{LT}_{\lambda x.M''}$ some P_1' exist such that $P_1 \stackrel{\tau}{\Longrightarrow} P_1' \in \mathcal{T}_{M''\{N/x\}}$.

(c) If $P \xrightarrow{n(p,l,r,v,f)} P'$, then $P' \in \mathcal{L}^{LT\cdot(n,l,r,v)}_{\lambda x.M'}$. Moreover for every closed λ-term $\lambda x.M''$ and every $P_1 \in \mathcal{L}^{LT}_{\lambda x.M''}$ some P_1' exists such that $P_1 \stackrel{n(p,l,r,v,f)}{\Longrightarrow} P_1' \in \mathcal{L}^{LT\cdot(n,l,r,v)}_{\lambda x.M''}$.

(d) P can and can only do either τ-actions or input actions at names in $I(TL)$.

Proof. Suppose the elements of $\mathcal{N}$ are enumerated by

$$u_0, u_1, \ldots, u_j, \ldots$$

and the elements of **LT**, the set of all the labeled trees, are enumerated by

$$LT_0, LT_1, \ldots, LT_k, \ldots.$$

Let h be a set of the sets of the π-processes that takes the following shape:

$$\{t_M, b_M, lb^u_M, lt^u_M, l^{LT}_M \mid M \in \Lambda^0, u \in \mathcal{N}, LT \in \mathbf{LT}\}.$$

An element of h is a set indexed either by a closed λ-term, or by a pair of a closed λ-term and a name, or by a pair of a closed λ-term and a labeled tree. The set of all such h is denoted by H. The function $F : H \to H$ is inductively defined as follows: For each

$$h = \{t_M, b_M, lb^u_M, lt^u_M, l^{LT}_M \mid M \in \Lambda^0, u \in \mathcal{N}, LT \in \mathbf{LT}\},$$

$F(h)$ is

$$\{t'_M, b'_M, lb'^u_M, lt'^u_M, l'^{LT}_M \mid M \in \Lambda^0, u \in \mathcal{N}, LT \in \mathbf{LT}\}$$

whose elements are constructed by the following inductions on all $M \in \Lambda^0$, $u \in \mathcal{N}$ and $LT \in \mathbf{LT}$.

1 $t'_M = t_M \cup \{[\![M]\!]_\lambda\}$, $b'_M = b_M$, $lt'^u_M = lt^u_M$, $lb'^u_M = lb^u_M$, and $l'^{LT}_M = l^{LT}_M$.

2 Suppose $P \in t_M$.

(a) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (s)(Sem \,|\, Q)$ for some Q and s, then $t'_M = t'_M \cup \{P'\}$.

(b) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (s)(Sem^- \,|\, Q)$ for some Q and s, and if there exists some Q' such that $Q \Longrightarrow Q' \not\rightarrow$ and $(s)(Sem^- \,|\, Q') \stackrel{\tau}{\longrightarrow} (s)(Sem \,|\, Q'') \equiv [\![M']\!]_\lambda$ for some M', then $b'_{M'} = b'_{M'} \cup \{P'\}$.

(c) If $M \equiv \lambda x.M'$ and $P \stackrel{\lambda u}{\longrightarrow} P'$, then $lt'^u_M = lt'^u_M \cup \{P'\}$.

3 Suppose $P \in b_M$.

(a) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (s)(Sem^- \,|\, Q)$ for some Q and s, then $b'_M = b'_M \cup \{P'\}$.

(b) If $P \stackrel{\tau}{\longrightarrow} P' \equiv (s)(Sem \,|\, Q)$ for some Q and s, then $t'_M = t'_M \cup \{P'\}$.

(c) If $M \equiv \lambda x.M'$ and $P \xrightarrow{\lambda u} P'$, then $lb'^u_M = lb'^u_M \cup \{P'\}$.

4 Suppose $P \in lt^u_M$.

(a) If $P \xrightarrow{\tau} P' \equiv (s)(Sem \,|\, Q)$ for some Q and s, then $lt'^u_M = lt'^u_M \cup \{P'\}$.

(b) If $P \xrightarrow{\tau} P' \equiv (s)(Sem^- \,|\, Q)$ and if there exist some P'', P''' such that $P' \xrightarrow{\tau} P'' \xrightarrow{u(p,l,r,v,f)} P'''$, then $l'^{\mathbf{1}_u}_M = l'^{\mathbf{1}_u}_M \cup \{P'\}$.

5 Suppose $P \in lb^u_M$.

(a) If $P \xrightarrow{\tau} P' \equiv (s)(Sem^- \,|\, Q)$ for some Q and s, then $lb'^u_M = lb'^u_M \cup \{P'\}$.

6 Suppose $P \in l^{LT}_M$.

(a) If $P \xrightarrow{\tau} P' \equiv (s)(Sem^- \,|\, Q)$ for some Q and s, then $l'^{LT}_M = l'^{LT}_M \cup \{P'\}$.

(b) If $P \xrightarrow{n(p,l,r,v,f)} P'$, then $l'^{LT\cdot(n,l,r,v)}_M = l'^{LT\cdot(n,l,r,v)}_M \cup \{P'\}$.

(c) If $P \xrightarrow{\tau} P' \equiv (s)(Sem \,|\, Q)$ for some Q and s, and if there exist some Q' and some closed λ-term N such that $Q \Longrightarrow Q' \not\rightarrow$ and $(s)(Sem \,|\, Q') \equiv [\![N]\!]_\lambda$, then $t'_N = t'_N \cup \{P'\}$.

By the above definition, the function F is monotone with respect to the subset relation. So we can construct an increasing sequence:

$$\begin{aligned} h^0 &\stackrel{\text{def}}{=} \{\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \cdots\}, \\ &\vdots \\ h^{i+1} &\stackrel{\text{def}}{=} F(h^i), \\ &\vdots \end{aligned}$$

The least fixed point h^ω is precisely the set

$$\{\mathcal{T}_M, \mathcal{B}_M, \mathcal{LB}^u_M, \mathcal{LT}^u_M, \mathcal{L}^{LT}_M \mid M \in \Lambda^0, u \in \mathcal{N}, LT \in \mathbf{LT}\}.$$

We denote the sets in h^i by $(t_M)^i, (b_M)^i, \cdots$. Each process P in $\mathcal{T}_M$ for example is in $(t_M)^k$ for some k. Intuitively $(t_M)^{i+1}, (b_M)^{i+1}, (lb^u_M)^{i+1}, (lt^u_M)^{i+1}, (l^{LT}_M)^{i+1}$ contain the processes that can be reached from $[\![M]\!]_\lambda$ after at most i actions.

We shall check that the lemma holds for $(t_M)^0$ and $(t_M)^1$. We then show that the lemma holds for $(t_M)^{i+1}$, for $i \geq 1$, under the assumption that it holds for $(t_M)^i$. The properties of the other sets can be checked similarly. To begin with, we make the following observations. These observations can all be proved by induction.

— All the processes in $\mathcal{T}_M$, $\mathcal{LT}^u_M$ are of the form $(s)(Sem \,|\, Q)$; and all the processes in $\mathcal{B}_M$, $\mathcal{LB}^u_M$ and $\mathcal{L}^{LT}_M$ are of the form $(s)(Sem^- \,|\, Q)$.

— If a process performs some action and evolves into a process in another set, then either it changes the state of the semaphore or it performs an input action.

— If M is not an abstraction term, then

$$\forall u \in \mathcal{N}.\forall LT \in \mathbf{LT}.(\mathcal{LT}^u_M = \mathcal{LB}^u_M = \mathcal{L}^{LT}_M = \emptyset).$$

In the following proof, we assume that M is an abstraction term when we write $\mathcal{LT}^u_M$, $\mathcal{LB}^u_M$ or $\mathcal{L}^{LT}_M$ for some u, LT.

— For each $P \in \mathcal{T}_M \cup \mathcal{B}_M$, after the reduction stage and all the replication stages are finished, it evolves into the encoding of M. Hence $P \Longrightarrow [\![M]\!]_\lambda$.

Since all the elements of h^0 are $\emptyset$, the lemma holds of h^0. For h^1, we need to check that the only element $[\![M]\!]_\lambda$ of $(t_M)^1$ satisfies the properties of the lemma. For each closed λ-term M, one has $[\![M]\!]_\lambda \equiv (s)(Sem \,|\, (\widetilde{vn})T_M^{n,\lambda,\perp})$, where $\widetilde{v} = \{v_x \mid x \text{ is free in } M)\}$, and $\widetilde{n}$ are the internal node names of the tree $T_M^{n,\lambda,\perp}$.

1 If $[\![M]\!]_\lambda \xrightarrow{\tau} P'$, since there are no interactions between any two components of $T_M^{n,\lambda,\perp}$, there must exist a redex node $L(P_i, n_i, m_i, m_i, v_i, \top) \in T_M^{n,\lambda,\perp}$ and

$$\begin{array}{rcl} L(p_i, n_i, m_i, m_i, v_i, \top) & \xrightarrow{s} & L_1, \\ Sem & \xrightarrow{\overline{s}} & Sem^-. \end{array}$$

Then P' is α-convertible to $(s)(Sem^- \,|\, Q)$ for some Q. Clearly $P' \notin \mathcal{T}_M$.

2 (a) If $[\![M]\!]_\lambda \xrightarrow{\tau} P'$, it has been shown above that there must be some β-redex in M. Without loss of generality, we may assume that $M \equiv C[(\lambda x.N)L]$ for some $C[\;], N, L$ and the node performing $\xrightarrow{s}$ is the redex node λx in $C[(\lambda x.N)L]$. We can rearrange the bound names in $T_M^{n,\lambda,\perp}$ by applying the structural congruence rules:

$$\begin{array}{rcl} T_M^{n,\lambda,\perp} & \equiv & T_{C[\text{-}]}^{n,\lambda,\perp} \,|\, L(n', p', l, r, \perp, f') \,|\, L(l, n', m, m, v_x, \top) \,|\, T_N^{m,l,\perp} \,|\, T_L^{r,n',\perp}, \\ P' & \equiv & (s)(\widetilde{v})(\widetilde{n})(Sem^- \,|\, T_{C[\text{-}]}^{n,\lambda,\perp} \,|\, L(n', p', l, r, \perp, f') \,|\, L_1 \,|\, T_N^{m,l,\perp} \,|\, T_L^{r,n',\perp}), \end{array}$$

where $L(l, n', m, m, v_x, \top) \xrightarrow{s} L_1$. Let Q' be defined by

$$Q' \stackrel{\text{def}}{=} (\widetilde{vn})(T_{C[\text{-}]}^{n,\lambda,\perp} \,|\, L(n', p', l, r, \perp, f') \,|\, L_1 \,|\, T_N^{m,l,\perp} \,|\, T_L^{r,n',\perp}).$$

After Q' has completed the reduction stage and the replication stage, it will evolve into

$$\begin{array}{rcl} Q' & \Longrightarrow & Q'' \\ & \equiv & (\widetilde{v'}\widetilde{n'})(T_{C[N\{L/x\}]}^{n,\lambda,\perp} \,|\, \overline{s} \,|\, [\![v_x := L]\!]_\perp) \\ & \equiv & (\widetilde{v''}\widetilde{n'})(T_{C[N\{L/x\}]}^{n,\lambda,\perp} \,|\, \overline{s}). \end{array}$$

Obviously Q'' has no τ-actions and $(s)(Sem^- \,|\, Q'') \xrightarrow{\tau} [\![C[N\{L/x\}]]\!]_\lambda$. Hence $M \to M' \equiv C[N\{L/x\}]$ and $P' \in (b_{M'})^2 \subseteq \mathcal{B}_{M'}$.

(b) If $M \equiv \lambda x.M'$ for some x, M', then

$$[\![\lambda x.M']\!]_\lambda \equiv (s)(Sem \,|\, (\widetilde{vn})(L(n, \lambda, m, m, v_x, \perp) \,|\, T_M^{m,n,\perp})).$$

Therefore $[\![M]\!]_\lambda$ can perform $\xrightarrow{\lambda u}$ and according to the construction we have $[\![M]\!]_\lambda \xrightarrow{\lambda u} P' \in (lt_M^u)^2 \subseteq \mathcal{LT}_M^u$. Moreover if $[\![M]\!]_\lambda \xrightarrow{\lambda u} P'$ then M must be in abstraction form and it follows from the construction that $P' \in (lt_M^u)^2 \subseteq \mathcal{LT}_M^u$.

(c) The encoding $[\![M]\!]_\lambda$ can either perform a τ-action to begin a simulation of β-reduction or, if M is an abstraction term, perform $\xrightarrow{\lambda u}$ for each u.

(d) If $M \to M'$, then similar to the case (a), there must exist some P' such that $[\![M]\!]_\lambda \xrightarrow{\tau} P' \in (b_{M'})^2 \subseteq \mathcal{B}_{M'}$.

So the lemma holds of h^1.

Now assume that the sets in h^i, for $i \geq 1$, hold of the properties of the lemma. Then for each closed λ-term M, consider the sets in h^{i+1}.

1 $(b_M)^{i+1}$. For each $P \in (b_M)^{i+1} \setminus (b_M)^i$, P must be of the form $(s)(Sem^- \,|\, Q)$ for some s, Q.

— By construction, there must be some P^* such that $P^* \xrightarrow{\tau} P$ and P^* is either in $(b_M)^i$ or in $(t_{M'})^i$ for some closed λ-term M'.

(a) If $P^* \in (b_M)^i$, then there does not exist an infinite τ-action sequence

$$P_0 \equiv P \xrightarrow{\tau} P_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_i \xrightarrow{\tau} \cdots$$

such that for any i, $P_i \in \mathcal{B}_M$, otherwise P^* would have an infinite τ-action sequence with all the processes in $\mathcal{B}_M$, contradicting to the induction hypothesis.

(b) If $P^* \in (t_{M'})^i$, then for some Q^*, $P^* \equiv (s)(Sem \,|\, Q^*)$ and $Q^* \xrightarrow{s} Q$. Moreover $M' \to M$. After performing a finite number of τ-actions Q will finish the reduction stage and the replication stage that simulate $M' \to M$. So if there was an infinite τ-action sequence

$$P \equiv P_0 \xrightarrow{\tau} P_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_i \xrightarrow{\tau} \cdots$$

such that for every i, $P_i \in \mathcal{B}_M$, then there would be an infinite τ-action sequence from P^*, contradicting to the induction hypothesis.

— If $P \xrightarrow{\tau} P'$, then there are two cases:

- The transition is caused by $Q \xrightarrow{\tau} Q'$. By construction,

$$P' \equiv (s)(Sem^- \,|\, Q') \in (b_M)^{i+2} \subseteq \mathcal{B}_M.$$

- The transition is induced by $Sem^- \xrightarrow{s} Sem$ and $Q \xrightarrow{\overline{s}} Q'$. By construction

$$P' \equiv (s)(Sem \,|\, Q') \in (t_M)^{i+2} \subseteq \mathcal{T}_M.$$

— If $M \equiv \lambda x.M'$ for some x, M' and $P \equiv (s)(Sem^- \,|\, Q)$, then the process Q can first perform the τ-action for a number of j times, for $j \geq 0$, to get the abstraction node λx ready. In other words there are some n, m and R such that

$$Q \underbrace{\xrightarrow{\tau} \cdots \xrightarrow{\tau}}_{j} Q' \equiv (nmv_x)(L(n, \lambda, m, m, v_x, \bot) \,|\, R).$$

Therefore

$$P \underbrace{\xrightarrow{\tau} \cdots \xrightarrow{\tau}}_{j} P' \equiv (s)(Sem^- \,|\, Q') \xrightarrow{\lambda u} P''.$$

It follows that $P' \in (b_M)^{i+j+1} \subseteq \mathcal{B}_M$ and $P'' \in (lb_M^u)^{i+j+2} \subseteq \mathcal{LB}_M^u$.

— If $P \xrightarrow{\lambda u} P'$, there must be an abstraction node with parent λ. By construction this is possible only if M is of the abstraction form. Thus $P' \in (lb_M^u)^{i+2} \subseteq \mathcal{LB}_M^u$ from the construction.

— The only public channel of P is λ. So P can only do either $\xrightarrow{\tau}$ or $\xrightarrow{\lambda u}$.

2 $(t_M)^{i+1}$. For each $P \in (t_M)^{i+1} \setminus (t_M)^i$, P must be in the form $(s)(Sem \,|\, Q)$ for some s, Q.

— By construction, there must be some P^* in $(t_M)^i$ or $(b_M)^i$, or in $(l_N^{LT})^i$ for some λ-term N and some labeled tree LT such that $P^* \stackrel{\tau}{\longrightarrow} P$.

(a) If $P^* \in (t_M)^i$, the case is easy.

(b) If $P^* \in b_M^i$, then $P^* \equiv (s)(Sem^- \,|\, \overline{s} \,|\, Q)$. If P has infinite τ-action sequence

$$(s)(Sem \,|\, Q) \stackrel{\tau}{\longrightarrow} (s)(Sem \,|\, Q_1) \stackrel{\tau}{\longrightarrow} \cdots$$

with all the processes in $\mathcal{T}_M$, then P^* also has an infinite τ-action sequence

$$(s)(Sem^- \,|\, \overline{s} \,|\, Q) \stackrel{\tau}{\longrightarrow} (s)(Sem^- \,|\, \overline{s} \,|\, Q_1) \stackrel{\tau}{\longrightarrow} \cdots$$

with all the processes belonging to $\mathcal{B}_M$, contradicting to the induction hypothesis.

(c) If $P^* \in (l_N^{LT})^i$, then some s, m, R and $\widetilde{n}$ exist such that

$$\begin{array}{rcl} P^* & \equiv & (s)(Sem^- \,|\, (\widetilde{n})(\overline{s}.L(m, \cdots) \,|\, R)), \\ P & \equiv & (s)(Sem \,|\, (\widetilde{n})(L(m, \cdots) \,|\, R). \end{array}$$

If P has an infinite τ-action sequence with all the processes in $\mathcal{T}_M$, then it must be caused by R because all interactions between $L(m, \cdots)$ and R should happen after turning off the semaphore. It follows that P^* would also have an infinite τ-action sequence with all the processes in $\mathcal{L}_M^{LT}$, contradicting to the induction hypothesis.

— Suppose $P \in (t_M)^{i+1} \setminus (t_M)^i$ and $P \stackrel{\tau}{\longrightarrow} P'$.

(a) If $P' \equiv (s)(Sem \,|\, Q')$ for some Q' with $Q \stackrel{\tau}{\longrightarrow} Q'$, then $P' \in (t_M)^{i+2} \subseteq \mathcal{T}_M$.

(b) If $P' \equiv (s)(Sem^- \,|\, Q')$ for some Q' with $Q \stackrel{s}{\longrightarrow} Q'$, there must exist some redex node, and M' such that $M \to M'$. As discussed in Section 5.1, after Q' completes this reduction stage and all the unfinished replication stages, it evolves into Q'' such that it has no more τ-actions and that $(s)(Sem^- \,|\, Q'') \stackrel{\tau}{\longrightarrow} [\![M']\!]_\lambda$. Therefore $P' \in (b_{M'})^{i+2} \subseteq \mathcal{B}_{M'}$.

— If $M \equiv \lambda x.M'$ for some x, M' and $P \equiv (s)(Sem \,|\, Q)$, then the process Q can first perform the τ-action j times, for $j \geq 0$, to get the abstraction node λx ready. In other words, there are some n, m and R such that

$$Q \underbrace{\stackrel{\tau}{\longrightarrow} \cdots \stackrel{\tau}{\longrightarrow}}_{j} Q' \equiv (nmv_x)(L(n, \lambda, m, m, v_x, \bot) \,|\, R).$$

So $P' \equiv (s)(Sem \,|\, Q') \stackrel{\lambda u}{\longrightarrow} P''$. By the construction, we have $P' \in (t_M)^{i+j+1} \subseteq \mathcal{T}_M$ and $P'' \in (lt_M^u)^{i+j+2} \subseteq \mathcal{LT}_M^u$.

— If $P \stackrel{\lambda u}{\longrightarrow} P'$, there must be an abstraction node with parent λ. So M must be of the abstraction form and we get $P' \in (lt_M^u)^{i+2} \subseteq \mathcal{LT}_M^u$ from the construction.

— The only public channel of P is λ. So P can only do either $\stackrel{\tau}{\longrightarrow}$ or $\stackrel{\lambda u}{\longrightarrow}$.

— As $P \in (t_M)^{i+1}$, P can evolve into the encoding of M by finishing all the unfinished replication stages. If $M \to M'$, then some P' exists such that $P \Longrightarrow [\![M]\!]_\lambda \stackrel{\tau}{\longrightarrow} P'$. Similar to the proof for h^1, one gets that $P' \in \mathcal{B}_{M'}$.

3 $(lb^u_M)^{i+1}$. There must exist some x, M_1 such that $M \equiv \lambda x.M_1$. By the construction, for each process P in $(lb^u_M)^{i+1} \setminus (lb^u_M)^i$, there exists some P^* in $(b_M)^i$ such that $P^* \stackrel{\lambda u}{\longrightarrow} P$, where P and P^* have the following forms:

$$\begin{array}{rcl} P^* & \equiv & (s)(Sem^- \,|\, (\widetilde{n})(L(n, \lambda, m, m, v_x, \bot) \,|\, R)), \\ P & \equiv & (s)(Sem^- \,|\, (\widetilde{n})(s.Q \,|\, R)), \end{array}$$

where $L(n, \lambda, m, m, v_x, \bot) \stackrel{\lambda u}{\longrightarrow} s.Q$ by definition.

— If P has an infinite τ-action sequence with all the processes in $\mathcal{LB}^u_M$, then it must be induced by an infinite τ-action sequence of R, which is contradictory to the induction hypothesis that P^* has no infinite τ-action sequence with all the processes in $\mathcal{B}_M$.

— If $P \stackrel{\tau}{\longrightarrow} P'$, then there are two cases: Either $R \stackrel{\tau}{\longrightarrow} R_1$ and

$$P' \equiv (s)(Sem^- \,|\, (\widetilde{n})(s.Q \,|\, R_1))$$

or $R \stackrel{\overline{s}}{\longrightarrow} R_2$ and $P' \equiv (s)(Sem \,|\, (\widetilde{n})(s.Q \,|\, R_2))$. In either case there is some Q^* such that $P^* \stackrel{\tau}{\longrightarrow} Q^* \stackrel{\lambda u}{\longrightarrow} P'$. Correspondingly, there are two cases for Q^*:

$$\begin{array}{l} Q^* \equiv (s)(Sem^- \,|\, (\widetilde{n})(L(n, \lambda, m, m, v_x, \bot) \,|\, R_1)), \\ Q^* \equiv (s)(Sem \,|\, (\widetilde{n})(L(n, \lambda, m, m, v_x, \bot) \,|\, R_2)). \end{array}$$

The process Q^* is in $(b_M)^{i+1}$ or $(t_M)^{i+1}$. It follows that

$$P' \in (lb^u_M)^{i+2} \cup (lt^u_M)^{i+2} \subseteq \mathcal{LB}^u_M \cup \mathcal{LT}^u_M.$$

— Since P has not finished the reduction stage, it can perform τ-actions. Because the input action on channel u is blocked by the semaphore, P can only perform τ-actions.

4 $(lt^u_M)^{i+1}$. There must exist some x, M_1 such that $M \equiv \lambda x.M_1$. If $P \in (lt^u_M)^{i+1} \setminus (lt^u_M)^i$, then by construction there exists $P^* \in (t_M)^i$ such that $P^* \stackrel{\lambda u}{\longrightarrow} P$. Here

$$\begin{array}{rcl} P^* & \equiv & (s)(Sem \,|\, (\widetilde{n})(L(n, \lambda, m, m, v_x, \bot) \,|\, R)), \\ P & \equiv & (s)(Sem \,|\, (\widetilde{n})(s.Q \,|\, R)), \end{array}$$

where $L(n, \lambda, m, m, v_x, \bot) \stackrel{\lambda u}{\longrightarrow} s.Q$ by definition.

— Similar to $(lb^u_M)^{i+1}$, for each process in $(lt^u_M)^{i+1}$, there does not exist an infinite τ-action sequence with all the processes in $\mathcal{LT}^u_M$.

— If $P \stackrel{\tau}{\longrightarrow} P'$, then P' is in one of three forms.

$$\begin{array}{rcl} P' & \equiv & (s)(Sem \,|\, (\widetilde{n})(s.Q \,|\, R_1)), \\ P' & \equiv & (s)(Sem^- \,|\, (\widetilde{n})(s.Q \,|\, R_2)), \\ P' & \equiv & (s)(Sem^- \,|(\widetilde{n})(Q \,|\, R)), \end{array}$$

where $R \xrightarrow{\tau} R_1$ and $R \xrightarrow{s} R_2$. If $P' \equiv (s)(Sem^- \,|\, (\widetilde{n})(Q \,|\, R))$, then there exist Q', R' such that

$$\begin{array}{lcl} Q & \xrightarrow{m(p_1,l_1,r_1,v_1,f_1)}\xrightarrow{u(p,l,r,v,f)} & Q', \\ R & \xrightarrow{\overline{m}\langle p_1,l_1,r_1,v_1,f_1\rangle} & R'. \end{array}$$

Therefore $P' \in (l_M^{\mathbf{1}_u})^{i+2} \subseteq \mathcal{L}_M^{\mathbf{1}_u}$. For the other two cases, there exists some Q^* such that $P^* \xrightarrow{\tau} Q^* \xrightarrow{\lambda u} P'$ as in case 3. Correspondingly, there are two cases for Q^*:

$$\begin{array}{lcl} Q^* & \equiv & (s)(Sem \,|\, (\widetilde{n})(L(n,\lambda,m,m,v_x,\bot) \,|\, R_1)), \\ Q^* & \equiv & (s)(Sem^- \,|\, (\widetilde{n})(L(n,\lambda,m,m,v_x,\bot) \,|\, R_2)). \end{array}$$

According to the argument we made in case 2, the process Q^* is in $(t_M)^{i+1}$ or there exists some M' such that $M \rightarrow M'$ and $Q^* \in (b_{M'})^{i+1}$. It follows that $P' \in (lt_M^u)^{i+2} \cup (lb_{M'}^u)^{i+2} \subseteq \mathcal{LT}_M^u \cup \mathcal{LB}_M^u$.

— Since $s.Q \xrightarrow{s}$, one has $P' \in (l_M^{\mathbf{1}_u})^{i+2} \subseteq \mathcal{L}_M^{\mathbf{1}_u}$ for some P'.

— Because the input action on channel u is blocked by the semaphore, P can only perform τ-actions.

5 $(l_M^{LT})^{i+1}$. $M \equiv \lambda x.M_1$ for some x, M_1. There are three major cases according to the types of LT:

(a) $LT = \mathbf{1}_u$ for some u and $P \in (l_M^{\mathbf{1}_u})^{i+1} \setminus (l_M^{\mathbf{1}_u})^i$.

— We first prove that there does not exist an infinite τ-action sequence with all the processes in $\mathcal{L}_M^{\mathbf{1}_u}$. First according to the construction there must be a P^* in $(lt_M^u)^i$ or in $(l_M^{\mathbf{1}_u})^i$ such that $P^* \longrightarrow P$. If $P^* \in (lt_M^u)^i$, then it must be of the form $(s)(Sem \,|\, (\widetilde{n})(s.m(\cdots).Q \,|\, R))$ for some $s, \widetilde{n}, Q, R$, where $L(n,\lambda,m,m,v_x,\bot) \xrightarrow{\lambda u} s.m(\cdots).Q$ and $n, m \in \widetilde{n}$. From the definition given in Fig. 4, the only action of Q is the input $u(p,l,r,v,f)$. So if P has an infinite τ-action sequence in which all the processes are in $\mathcal{L}_M^{\mathbf{1}_u}$, then P^* would also have and infinite τ-action sequence with all the processes in $\mathcal{LT}_M^u$, which is contradictory to the induction hypothesis.

— If $P^* \in (l_M^{\mathbf{1}_u})^i$ and $P \xrightarrow{\tau} P'$, then since the semaphore will not be turned off until the importation of a λ-term is finished, P' must be in the form $(s)(Sem^- \,|\, Q)$. By construction we have $P' \in (l_M^{\mathbf{1}_u})^{i+2} \subseteq \mathcal{L}_M^{\mathbf{1}_u}$.

— If $P^* \in (l_M^{\mathbf{1}_u})^i$ and $P \xrightarrow{u(p,l,r,v,f)} P'$, then by the construction and the definition given in Fig. 4, one has $P' \in (l_M^{\mathbf{1}_u\cdot(u,l,r,v)})^{i+2} \subseteq \mathcal{L}_M^{\mathbf{1}_u\cdot(u,l,r,v)}$. For every λ-term M' of the form $\lambda x.M''$ and every process $O \in (l_{\lambda x.M''}^{\mathbf{1}_u})^{i+1}$, according to the construction, there is some $O^* \in (l_M^{\mathbf{1}_u})^i$ such that $O^* \Longrightarrow O$ and O^* is of the form $(s)(Sem^- \,|\, (\widetilde{n})(m(\cdots).Q \,|\, R))$ for some $s, \widetilde{n}, Q, R$, where

$$L(n,\lambda,m,m,v_x,\bot) \xrightarrow{\lambda u}\xrightarrow{s} m(\cdots).Q.$$

Since Q can perform an input on channel u, say $Q \xrightarrow{u(p,l,r,v,f)} Q'$, there must

exist some k, O' such that

$$O \underbrace{\xrightarrow{\tau} \cdots \xrightarrow{\tau}}_{k} \xrightarrow{u(p,l,r,v,f)} O',$$

and consequently $O' \in (l_{\lambda x.M''}^{\mathbf{1}_u \cdot (u,l,r,v)})^{i+k+1} \subseteq \mathcal{L}_{\lambda x.M''}^{\mathbf{1}_u \cdot (u,l,r,v)}$.

— As discussed above the process P can do either a τ action or an input at name u. There are no other public channels in any process $P \in (l_{\lambda x.M'}^{\mathbf{1}_u})^{i+1}$. Since $I(\mathbf{1}_u) = \{u\}$, we get that the process P can and can only perform either τ-actions or input actions at the name in $I(\mathbf{1}_u)$.

(b) $I(LT) \neq \emptyset$ and $P \in (l_M^{LT})^{i+1} \setminus (l_M^{LT})^i$.

— We prove that there does not exist an infinite τ-action sequence with all the processes in $\mathcal{L}_M^{LT}$. First according to the construction, there must be a P^* such that either $P^* \in (l_M^{LT})^i$ and $P^* \xrightarrow{\tau} P$, or $P^* \in (l_M^{LT'})^i$ and $P^* \xrightarrow{n(p,l,r,v,f)} P$ and $LT = LT' \cdot (n, l, r, v)$. The case of $P^* \in (l_M^{LT})^i$ is easy. If $P^* \in (l_M^{LT'})^i$, then since $I(LT) \neq \emptyset$, P has not finished the importation. The new input on P^* will not introduce infinite τ-actions, since if P had an infinite τ-action sequence in which all the processes are in $\mathcal{L}_M^{LT}$, then P^* would also have an infinite τ-action sequence with all the processes in $\mathcal{L}_M^{LT'}$, contradicting to the induction hypothesis.

— If $P \xrightarrow{\tau} P'$, then since the semaphore will not be turned off until the importation of a λ-term is finished, P' must be in the form $(s)(Sem^- \,|\, Q)$. By construction one has $P' \in (l_M^{LT})^{i+2} \subseteq \mathcal{L}_M^{LT}$.

— Since the process P has not finished the importation, it will continue to read from the environment. The set of the public channels in P is $I(LT)$, meaning that if $P \xrightarrow{n(p,l,r,v,f)} P'$ then $n \in I(LT)$. According to the construction, $P' \in (l_M^{LT \cdot (n,l,r,v)})^{i+2} \subseteq \mathcal{L}_M^{LT \cdot (n,l,r,v)}$. For every closed λ-term M' of the form $\lambda x.M''$ and every process $O \in (l_{\lambda x.M''}^{LT})^{i+1}$, O can perform an input action at a name n if and only if $n \in I(LT)$. So we must also have

$$O \underbrace{\xrightarrow{\tau} \cdots \xrightarrow{\tau}}_{k} \xrightarrow{n(p,l,r,v,f)} O'$$

and therefore $O' \in (l_{\lambda x.M''}^{LT \cdot (n,l,r,v)})^{i+k+1} \subseteq \mathcal{L}_{\lambda x.M''}^{LT \cdot (n,l,r,v)}$.

— As just said, the process P can do either a τ-action or an input at names in $I(LT)$. There are no other public channels in any process $P \in (l_{\lambda x.M'}^{LT})^{i+1}$. So the process P can and can only perform either τ-actions or input actions at names in $I(LT)$.

(c) If $I(LT) = \emptyset$, the process $P \in (l_M^{LT})^{i+1} \setminus (l_M^{LT})^i$ has completed the importation from the environment and LT must be a labeled tree of some λ-term N.

— First we argue that there does not exist an infinite τ-action sequence with all the processes in $\mathcal{L}_M^{LT}$. According to the construction, there must be a P^* such that either $P^* \in (l_M^{LT})^i$ and $P^* \xrightarrow{\tau} P$, or $P^* \in (l_M^{LT'})^i$ and $P^* \xrightarrow{n(p,l,r,v,f)} P$

and $LT = LT' \cdot (n, l, r, v)$. The case of $P^* \in (l_M^{LT})^i$ is easy. If $P^* \in (l_M^{LT'})^i$, then since P has finished the importation, after at most finite steps of reduction and replication, it would have no more τ-actions except those turning off the semaphore. Otherwise P^* would have an infinite τ-action sequence with all the processes in $\mathcal{L}_M^{LT'}$, contradicting to the induction hypothesis.

— If $P \equiv (s)(Sem^- \mid Q)$ and $P \xrightarrow{\tau} P'$, there are two cases.

  – The transition is caused by $Q \xrightarrow{\tau} Q'$ and $P' \equiv (s)(Sem^- \mid Q')$. Then $P' \in (l_M^{LT})^{i+2} \subseteq \mathcal{L}_M^{LT}$.

  – The transition is caused by $Q \xrightarrow{\overline{s}} Q'$ and $P' \equiv (s)(Sem \mid Q')$. According to the definition of the encoding in Fig. 4 and the discussion in Section 5.2, the encoding of some closed λ-term N' has been imported at this stage. Then after Q' completes the reduction stage and all the replication stages, it would evolve into some Q'' such that $Q' \stackrel{\tau}{\Longrightarrow} Q'' \not\rightarrow$. Then $(s)(Sem \mid Q') \equiv [\![M'\{N'/x\}]\!]_\lambda$ and $P' \in (t_{M'\{N'/x\}})^{i+2} \subseteq \mathcal{T}_{M'\{N'/x\}}$, where M' is such that $M \equiv \lambda x.M'$. For every closed λ-term $\lambda x.M''$ and every process $O \in (l_{\lambda x.M''}^{LT})^{i+1}$, O must also have finished the importation. Thus $O \stackrel{\tau}{\Longrightarrow} O' \in \mathcal{T}_{M''\{N'/x\}}$ for some O'.

— P can perform τ-actions since the semaphore has not been turned off. In the definition of $RAbs(n, \lambda, m, v_x, f)$, the only public channel λ in P is blocked by $\overline{s}$. So P can only perform τ-actions.

This is the end of the case analysis. □

A.2. *Proof of Lemma 6*

First of all notice that (2) can be deduced from (1), (3a), (3b), (4a), (4b), (5a), (5b) of Lemma 6 and (1) of Lemma 4. So we only need to prove (1, 3-5) of Lemma 6. Here is the case analysis:

1 $P \in \mathcal{CT}_{\lambda x.M}^N$. Then $P \equiv (u)(Q \mid T(u, N))$ for some u, where $Q \in \mathcal{LT}_{\lambda x.M}^u$.

— The interactions between Q and $T(u, N)$ can only contribute a finite number of τ-actions. If P had an infinite τ-action sequence with all the processes in $\mathcal{CT}_{\lambda x.M}^N$, then Q would have an infinite τ-action sequence with all the processes in $\mathcal{LT}_{\lambda x.M}^u$, contradicting to (1) of Lemma 5.

— If $P \xrightarrow{\tau} P'$, then the transition must be caused by some τ-action of Q, because $T(u, N)$ has no τ-actions and the interaction between Q and $T(u, N)$ cannot happen right now. Assume $P' \stackrel{\text{def}}{=} (u)(Q' \mid T(u, N))$, where $Q \xrightarrow{\tau} Q'$. By (4a) of Lemma 5, $Q' \in \mathcal{LT}_{\lambda x.M}^u \cup \mathcal{L}_{\lambda x.M}^{\mathbf{1}_u}$, or $\exists M'.M \to M' \wedge P' \in \mathcal{LB}_{\lambda x.M'}^u$. Accordingly, one has $P' \in \mathcal{CT}_{\lambda x.M}^N \cup \mathcal{C}_{\lambda x.M}^N$, or $\exists M'.M \to M' \wedge P' \in \mathcal{CB}_{\lambda x.M'}^N$.

— By (4b) of Lemma 5, there exists Q' such that $Q \xrightarrow{\tau} Q' \in \mathcal{L}_{\lambda x.M}^{\mathbf{1}_u}$. Therefore some P' exists such that $P \xrightarrow{\tau} P' \in \mathcal{C}_{\lambda x.M}^N \subseteq \mathcal{CT}_{\lambda x.M}^N \cup \mathcal{C}_{\lambda x.M}^N$.

— It follows from (4c) of Lemma 5 that Q can and can only do τ-actions. The process $T(u, N)$ has only output actions at private channels. An interaction between Q,

and its descendants, and $T(u,N)$ cannot happen before the process jumps out of $\mathcal{CT}^N_{\lambda x.M}$. Therefore P can only perform τ-actions.

2 $P \in \mathcal{CB}^N_{\lambda x.M}$. Then $P \equiv (u)(Q \,|\, T(u,N))$, where $Q \in \mathcal{LB}^u_{\lambda x.M}$. The following arguments are very much like those in the previous case.

— If P has an infinite τ-action sequence with all the processes in $\mathcal{CT}^N_{\lambda x.M}$, then Q would have an infinite τ-action sequence with all the processes in $\mathcal{LB}^u_{\lambda x.M}$, contradicting to (1) of Lemma 5.

— If $P \xrightarrow{\tau} P'$, then this transition must be caused by some τ-action of Q. One can assume $P' \stackrel{\text{def}}{=} (u)(Q' \,|\, T(u,N))$, where $Q \xrightarrow{\tau} Q'$. By (5a) of Lemma 5, $Q' \in \mathcal{LB}^u_{\lambda x.M} \cup \mathcal{LT}^u_{\lambda x.M}$. Consequently $P' \in \mathcal{CB}^N_{\lambda x.M} \cup \mathcal{CT}^N_{\lambda x.M}$.

— It follows from (5b) of Lemma 5 that Q can and can only do τ-actions. The process $T(u,N)$ has only output actions at private channels. An interaction between Q, and its descendants, and $T(u,N)$ cannot happen before the process jumps out of $\mathcal{CT}^N_{\lambda x.M}$. Therefore P can only perform τ-actions.

3 $P \in \mathcal{C}^N_{\lambda x.M}$. There are two cases:

— $P \equiv (u)(Q \,|\, T(u,N))$ where $Q \in \mathcal{L}^{\mathbf{1}_u}_{\lambda x.M}$. By Lemma 5, Q has no infinite τ-action sequences. So an infinite τ-action sequence of P must be caused by the interactions between Q and $T(u,N)$. Without loss of generality, we may assume that $T(u,N)$ has the following transitions:

$$T(u,N) \xrightarrow{\overline{u}\langle p,l,r,v,f\rangle} \xrightarrow{\overline{l}\langle p',l',r',v',f'\rangle} \cdots \xrightarrow{\overline{n}\langle p_n,\perp,\perp,v_n,f_n\rangle} \mathbf{0}.$$

Correspondingly Q interacts in the following manner:

$$Q \underbrace{\Longrightarrow Q_1}_{\mathcal{L}^{\mathbf{1}_u}_{\lambda x.M}} \xrightarrow{u(p,l,r,v,f)} \underbrace{\Longrightarrow Q_2}_{\mathcal{L}^{\mathbf{1}_u\cdot(u,l,r,v)}_{\lambda x.M}} \cdots Q_{n-1} \xrightarrow{n(p_n,\perp,\perp,v_n,f_n)} \underbrace{\Longrightarrow Q_n}_{\mathcal{L}^{LT}_{\lambda x.M}} \underbrace{\Longrightarrow Q_{n+1}}_{\mathcal{T}_{M\{N/x\}}}.$$

By (1) and (6) of Lemma 5, the τ-actions between Q and Q_{n+1} are finite. Meanwhile $T(u,N)$ can only do a finite number of actions. So P does not have any infinite τ-action sequences in $\mathcal{C}^N_{\lambda x.M}$. Another possibility is that there exists some $P^* \equiv (u)(Q_0 \,|\, T(u,N)) \in \mathcal{C}^N_{\lambda x.M}$ such that $P^* \stackrel{\tau}{\Longrightarrow} P$. Since P^* does not have any infinite τ-action sequences with all the processes in $\mathcal{C}^N_{\lambda x.M}$, neither does P.

— According to the above discussion, a process in $\mathcal{C}^N_{\lambda x.M}$ is of the following form

$$P \equiv (\widetilde{u_i})(Q_i \,|\, \Pi_{k\in\{1,2,\cdots,n_i\}} T(u_{i_k}, N_{i_k})),$$

where $N_{i_1}, N_{i_2}, \cdots, N_{i_{n_i}}$ are sub-terms of N, $Q_i \in \mathcal{L}^{LT_i}_{\lambda x.M}$, and

$$I(LT_i) = \widetilde{u_i} = \{u_{i_1}, u_{i_2}, \cdots, u_{i_{n_i}}\}.$$

Let LT_N be the labeled tree of the λ-term N. Then $LT_i \Subset LT_N$. If $P \xrightarrow{\tau} P'$, then there are two subcases:

– The transition is induced by $Q_i \xrightarrow{\tau} Q'_i$. It follows from (6a) of Lemma 5 that either $Q'_i \in \mathcal{L}^{LT_i}_{\lambda x.M}$ or $Q'_i \in \mathcal{T}_{M\{N'/x\}}$ for some closed λ-term N'. If $Q'_i \in \mathcal{L}^{LT_i}_{\lambda x.M}$, then clearly $P' \in \mathcal{C}^N_{\lambda x.M}$. In the case that $Q'_i \in \mathcal{T}_{M\{N'/x\}}$ for some closed λ-term N', we know from the proof of Lemma 8 that $I(LT_i)$ must be $\emptyset$. As

$LT_i \Subset LT_N$ and $I(LT_i) = \emptyset$, we have $LT_i = LT_N$. Consequently $N' \equiv N$ and $P' \equiv Q_i' \in \mathcal{T}_{M\{N/x\}}$.

– The transition is induced by an interaction between Q_i and $T(u_{i_k}, N_{i_k})$ for some $i_k \in \{1, 2, \cdots, n_i\}$. Then

$$Q_i \xrightarrow{u_{i_k}(p,l,r,v,f)} Q_j$$

and

$$T(u_{i_k}, N_{i_k}) \xrightarrow{\overline{u_{i_k}}\langle p,l,r,v,f\rangle} T',$$

where T' is of the shape $\Pi_{k\in\{1,2,\cdots,n_j\}} T(u_{j_k}, N_{j_k})$. Therefore $P \xrightarrow{\tau} P' \equiv (\widetilde{u_j})(Q_j \,|\, T')$. By (6c) of Lemma 5, the process Q_j is in $\mathcal{L}^{LT_j}_{\lambda x.M}$ where $LT_j = LT_i \cdot (u_{i_k}, l, r, v)$ and $\widetilde{u_j} = I(LT_j)$. So we have $P' \in \mathcal{C}^N_{\lambda x.M}$.

— By (6d) of Lemma 5, Q_i can and can only do either τ-actions or input actions at names in $I(LT_i)$. As names in $I(LT_i)$ have been restricted in P, P can and can only perform τ-actions.

We are done.

A.3. *Proof of Lemma 7*

Recall that in the proof of Proposition 1 we have constructed three relations $\mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{R}_3$. Now define $\mathcal{R}_1'$ to be the following relation

$$\left\{ (P,Q) \;\middle|\; \begin{array}{l} \exists M,N \in \Lambda^0.(M =_a N \\ \wedge\ P \in \mathcal{T}_M \cup \mathcal{B}_M \cup (\bigcup_{L\{L'/x\}\equiv M} \mathcal{CT}^{L'}_{\lambda x.L} \cup \mathcal{CB}^{L'}_{\lambda x.L} \cup \mathcal{C}^{L'}_{\lambda x.L}) \\ \wedge\ Q \in \mathcal{T}_N \cup \mathcal{B}_N \cup (\bigcup_{L\{L'/x\}\equiv N} \mathcal{CT}^{L'}_{\lambda x.L} \cup \mathcal{CB}^{L'}_{\lambda x.L} \cup \mathcal{C}^{L'}_{\lambda x.L})) \end{array} \right\}.$$

So if

$$P, Q \in \mathcal{T}_M \cup \mathcal{B}_M \cup (\bigcup_{L\{L'/x\}\equiv M} \mathcal{CT}^{L'}_{\lambda x.L} \cup \mathcal{CB}^{L'}_{\lambda x.L} \cup \mathcal{C}^{L'}_{\lambda x.L}),$$

then $(P,Q) \in \mathcal{R}_1'$ due to $M =_a M$. Similar to Proposition 1, we need to check that the elements in $\mathcal{R} \stackrel{\text{def}}{=} \mathcal{R}_1' \cup \mathcal{R}_2 \cup \mathcal{R}_3$ satisfy the following bisimulation property:

1 If $Q\mathcal{R}P \xrightarrow{\tau} P'$ then $\exists Q'.Q \Longrightarrow Q'\mathcal{R}P'$.
2 If $Q\mathcal{R}P \xrightarrow{a(\widetilde{n})} P'$ then $\exists Q'.Q \stackrel{a(\widetilde{n})}{\Longrightarrow} Q'\mathcal{R}P'$.

Since $\mathcal{R}_1 \subseteq \mathcal{R}_1'$, we only need to check for the pairs (P,Q) in $\mathcal{R}_1' \setminus \mathcal{R}_1$.

— Suppose $M, N \in \Lambda^0$, $\lambda x.L, L' \in \Lambda^0$, $M =_a N$, $N \equiv L\{L'/x\}$, $P \in \mathcal{T}_M$ and $Q \in \mathcal{CT}^{L'}_{\lambda x.L} \cup \mathcal{CB}^{L'}_{\lambda x.L} \cup \mathcal{C}^{L'}_{\lambda x.L}$. By (2) of Lemma 5, there are three cases.

1 If $P \xrightarrow{\tau} P' \in \mathcal{T}_M$, then $P'\mathcal{R}Q$.
2 If $P \xrightarrow{\tau} P'$ and $\exists M'.M \to M' \wedge P' \in \mathcal{B}_{M'}$, then $P'\mathcal{R}Q$ as $M' =_a M =_a N$.
3 If $P \xrightarrow{\lambda u} P' \in \mathcal{LT}^u_M$, then by (2b) of Lemma 5 there exist x, M' such that $M \equiv \lambda x.M'$. Since $M =_a N$, we must have $N \to^* \lambda x.N'$ and $M =_a \lambda x.N'$. According to (2) of Lemma 6, one has that $Q \Longrightarrow Q' \in \mathcal{T}_{\lambda x.N'}$. It follows from (2b) of Lemma 5 that $Q' \Longrightarrow\xrightarrow{\lambda u} Q'' \in \mathcal{LT}^u_{\lambda x.N'}$. So $Q \stackrel{\lambda u}{\Longrightarrow} Q''$ and $P'\mathcal{R}Q''$.

— Suppose $M, N \in \Lambda^0$, $\lambda x.L, L' \in \Lambda^0$, $M =_a N$, $N \equiv L\{L'/x\}$, $P \in \mathcal{B}_M$ and $Q \in \mathcal{CT}^{L'}_{\lambda x.L} \cup \mathcal{CB}^{L'}_{\lambda x.L} \cup \mathcal{C}^{L'}_{\lambda x.L}$. According to (3) of Lemma 5, there are three cases.

1 If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{B}_M$, then $P'\mathcal{R}Q$.
2 If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{T}_M$, then $P'\mathcal{R}Q$.
3 If $P \stackrel{\lambda u}{\longrightarrow} P' \in \mathcal{LB}^u_M$, then similarly $Q' \Longrightarrow\stackrel{\lambda u}{\longrightarrow} Q'' \in \mathcal{LT}^u_{\lambda x.N'}$, where $N \rightarrow^* \lambda x.N'$. Hence $P'\mathcal{R}Q''$.

— Suppose $M, N \in \Lambda^0$, $\lambda x.L, L' \in \Lambda^0$, $M =_a N$, $M \equiv L\{L'/x\}$, $P \in \mathcal{CT}^{L'}_{\lambda x.L}$ and $Q \in \mathcal{T}_N \cup \mathcal{B}_N$. By (3) of Lemma 6, there are three cases.

1 If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{CT}^{L'}_{\lambda x.L}$, then $P'\mathcal{R}Q$.
2 If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{C}^{L'}_{\lambda x.L}$, then $P'\mathcal{R}Q$.
3 If $P \stackrel{\tau}{\longrightarrow} P'$ and $\exists L''.L \rightarrow L'' \wedge P' \in \mathcal{CB}^{L'}_{\lambda x.L''}$, then $P'\mathcal{R}Q$ since $L''\{L'/x\} =_a L\{L'/x\} =_a N$.

— Suppose $M, N \in \Lambda^0$, $\lambda x.L, L' \in \Lambda^0$, $M =_a N$, $M \equiv L\{L'/x\}$, $P \in \mathcal{CB}^{L'}_{\lambda x.L}$ and $Q \in \mathcal{T}_N \cup \mathcal{B}_N$. By (4) of Lemma 6, there are two cases.

1 If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{CB}^{L'}_{\lambda x.L}$, then $P'\mathcal{R}Q$.
2 If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{CT}^{L'}_{\lambda x.L}$, then $P'\mathcal{R}Q$.

— Suppose $M, N \in \Lambda^0$, $\lambda x.L, L' \in \Lambda^0$, $M =_a N$, $M \equiv L\{L'/x\}$, $P \in \mathcal{C}^{L'}_{\lambda x.L}$ and $Q \in \mathcal{T}_N \cup \mathcal{B}_N$. By (5) of Lemma 6 there are two cases.

1 If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{C}^{L'}_{\lambda x.L}$, then $P'\mathcal{R}Q$;
2 If $P \stackrel{\tau}{\longrightarrow} P' \in \mathcal{T}_{L\{L'/x\}}$, then $P'\mathcal{R}Q$ since $L\{L'/x\} \equiv M =_a N$.

Conclude that for every $M \in \Lambda^0$ and all P, Q in

$$\mathcal{T}_M \cup \mathcal{B}_M \cup (\bigcup_{L\{N/x\} \equiv M} \mathcal{CT}^N_{\lambda x.L} \cup \mathcal{CB}^N_{\lambda x.L} \cup \mathcal{C}^N_{\lambda x.L}),$$

one has that $P \approx Q$.