Nominal Structural Operational Semantics*

Luca Aceto[1], Matteo Cimini[2], Anna Ingolfsdottir[1], MohammadReza Mousavi[3], Michel A. Reniers[4], and Murdoch J. Gabbay[5]

[1] ICE-TCS, School of Computer Science, Reykjavík University, Menntavegur 1, IS-101, Reykjavík, Iceland.
[2] Center for Research in Extreme Scale Technologies, Indiana University, 420 North Walnut Street, Bloomington, IN, 47404, USA
[3] Center for Research on Embedded Systems, School of Information Science, Halmstad University, Sweden
[4] Department of Mechanical Engineering, Eindhoven University of Technology, P.O. Box 513, NL-5600MB, Eindhoven, The Netherlands
[5] Computer Science Department, Heriot-Watt University, Riccarton, EH14 4AS Edinburgh, United Kingdom.

Abstract. Plotkin's style of Structural Operational Semantics (SOS) has become a de facto standard in giving operational semantics to formalisms and process calculi. In many such formalisms and calculi, the concepts of names, variables and binders are essential ingredients. In this paper, we propose a formal framework for dealing with names in SOS. The framework is based on the Nominal Logic of Gabbay and Pitts and hence is called Nominal SOS. We define nominal bisimilarity, an adaptation of the notion of bisimilarity that is aware of binding. We provide evidence of the expressiveness of the framework by formulating the early π-calculus and Abramsky's lazy λ-calculus within Nominal SOS. For both calculi we establish the operational correspondence with the original calculi. Moreover, in the context of the π-calculus, we prove that nominal bisimilarity coincides with Sangiorgi's open bisimilarity and in the context of the λ-calculus we prove that nominal bisimilarity coincides with Abramsky's applicative bisimilarity.

1 Introduction

The development of a formal semantics for programming and specification languages is a necessary first step towards rigorous reasoning about them. For instance, a formal semantics allows one to prove the correctness of language

* This paper is a considerably revised and expanded version of the conference article [11]. The research presented in this paper has been supported by the project 'Nominal Structural Operational Semantics' (nr. 141558-051) of the Icelandic Research Fund. Luca Aceto and Anna Ingolfsdottir also acknowledge the support of the grant 'Formal Methods for the Development and Evaluation of Sustainable Systems', Programme NILS Science and Sustainability, Priority Sectors Programme of the EEA Grant Framework.

implementations, and is a prerequisite for proving the validity of program optimizations. Operational semantics is a widely-used methodology to define formal semantics for computer languages, which represents the execution of programs as step-by-step development of an abstract machine. *Structural Operational Semantics* (SOS) was introduced by Gordon Plotkin in [42], reprinted in [21], as a logical and structural approach to defining operational semantics. The logical structure of SOS specifications supports a variety of reasoning principles that can be used to prove properties of programs whose semantics is given using SOS. Moreover, SOS language specifications can be used for rapid prototyping of language designs and to provide experimental implementations of computer languages.

Thanks to its intuitive appeal and flexibility, SOS has become the de facto standard for defining operational semantics, and a wealth of programming and executable specification languages have been given formal semantics using it. In recent years much work on the underlying theory as well as on the practice of SOS has been carried out—see, e.g., [3, 38] and [10, 24, 37], respectively. However, a substantial amount of work remains to be done in this rapidly-evolving area of research. This paper will focus on one of the crucial aspects in the definition of semantic models for programming and specification languages that has so far received relatively little attention in the literature of the meta-theory of SOS, i.e., the treatment of concepts such as variables, names and binders.

Many programming and specification languages make use of the concepts of names and binders. For example, in the π-calculus [35, 36, 48], names are first-class objects and the whole language is built on the idea that concurrent agents communicate by exchanging names. Binders are syntactic constructs that are used to scope the use of names in expressions. Examples of binders are input-prefixing operators, recursion combinators, restriction operators, infinite-sum operators and the time-integration operator [5, 34, 22, 48].

In this paper we propose a formal framework for the handling of names in SOS, called *Nominal SOS*, which is based on the nominal techniques of Gabbay, Pitts, and Urban [16, 53].

In the semantics of most nominal calculi some basic notions such as α-conversion and substitution are used. We show that these notions can be naturally captured in the Nominal SOS framework. Moreover, we specify two of the most prominent examples of nominal calculi, namely the lazy λ-calculus and the early π-calculus, in Nominal SOS and show that our specifications coincide operationally with the original definitions of [2] and [48], respectively. Finally, we define a notion of nominal bisimilarity naturally arising from our framework. We show that in the case of the π-calculus our notion coincides with the well-known open bisimilarity, [48, 47]. On the other hand, we prove that nominal bisimilarity is not a satisfactory notion of equivalence over the lazy λ-calculus. However, one of the most interesting notions of bisimilarity in the context of the λ-calculus is the *applicative bisimilarity* due to Samson Abramsky, [2]. We define this notion of equivalence in the framework of Nominal SOS and prove that coincides with its original counter-part. We finally give an alternative formulation of the lazy

λ-calculus and we prove that in the context of this formulation, nominal and applicative bisimilarity coincide.

The reader must bear in mind that this paper should be considered as containing the basic developments of the framework of Nominal SOS, accompanied by some examples of its application. Refinements and extensions of the framework are possible and left as future work. The investigation of the theory of Nominal SOS is also part of our future research plans. The reader may find in Section 9 an outline of our plans for future research. The overarching aim is to develop the framework of Nominal SOS in a way that is comparable to that of the standard theory of SOS, as surveyed in, e.g., [3, 38], and to hopefully establish Nominal SOS as a framework of reference for the study and the development of the theory of languages with first-class notions of names and binders.

Nominal SOS is not the only approach studied so far in the literature that aims at a uniform treatment of binders and names in the operational semantics of programming and specification languages. We are aware of a number of existing approaches that accommodate variables and binders inside variations on the SOS framework, and we discuss the most relevant approaches in Section 8.

Structure of the paper. The rest of this paper is organized as follows. In Section 2 we define nominal terms as used in the rest of the paper. In Section 3 we define Nominal SOS, an SOS framework extended with names and binders. We show in Section 4 how α-conversion and different types of substitutions can be accommodated in the Nominal SOS framework. In Section 5, we give the definitions of the λ- and the π-calculus in our framework and show their correspondence with the original presentations. In Section 6, we define the notion of nominal bisimilarity and show that coincides with open bisimilarity over the early π-calculus. In Section 7, we formulate the notion of applicative bisimilarity in the framework of Nominal SOS and show that it coincides with the original notion of Abramsky. Section 8 discusses related works in some detail and Section 9 concludes the paper by pointing out some directions for future work. For the sake of readability, proofs of some results and some technical definitions are collected in a series of sections that follow Section 9.

2 Nominal terms

The following definitions of *sorts* and *nominal signature* are familiar from [53].

Definition 1 (Sorts) *We assume a set of* atom sorts *and a disjoint set of* base sorts *(or* sorts of data*).* Sorts *are defined by the following grammar:*

$$\sigma ::= \mathbf{1} \mid \delta \mid A \mid [A]\sigma \mid \sigma \times \sigma,$$

where $\mathbf{1}$ *is the unit sort,* δ *is a base sort,* A *is an atom sort, and* $\times$ *denotes pairing.*

In the above-given grammar $[A]\sigma$ denotes an abstraction sort. Intuitively, $[A]\sigma$ is a sort whose elements are functions from objects of sort A to objects of sort

σ. As is standard, pair sorts will associate to the left, so that $\sigma_1 \times \sigma_2 \times \sigma_3$ stands for $(\sigma_1 \times \sigma_2) \times \sigma_3$.

Definition 2 (Nominal Signature) *A* nominal signature *(or simply* signature*)* Σ *is a triple* $(\Delta, \mathbb{A}, F)$*, where*

1. Δ *is a set of* base sorts *ranged over by* δ*,*
2. $\mathbb{A}$ *is a set of* atom sorts *ranged over by* A*, and*
3. F *is a set of* operators $f_{(\sigma_1 \times \ldots \times \sigma_n) \to \delta}$*, denoting a function symbol* f *with* arity $(\sigma_1 \times \ldots \times \sigma_n) \to \delta$*, where* $n \geq 0$*.*

By way of example, a function symbol of arity $(\sigma_1 \times \sigma_2 \times \sigma_3) \to \delta$ can be applied to three arguments of sorts σ_1, σ_2 and σ_3, respectively, and the term resulting from this application is of sort δ.

For each atom sort A, we fix a countably infinite set of *atoms* a_A, b_A, c_A, d_A, n_A, m_A and, for each sort σ, we assume a countably infinite set V_σ of *variable symbols* x_σ, y_σ, z_σ. All the above sets are pairwise disjoint.

We will mostly write just f, a, b, c, d, n, m, and x, y, z, leaving arities and sorts implicit (but still present).

Definition 3 (Nominal Terms) *Given a signature* $\Sigma = (\Delta, \mathbb{A}, F)$*, the set of* nominal terms *over the signature* Σ *is denoted by* $\mathbb{T}(\Sigma)$ *and it is defined as follows, where we write* t_σ *for a term* t *of sort* σ*:*

$$t, u ::= \langle\rangle_{\mathbf{1}} \mid x_\sigma \mid a_A \mid ([a_A]t_\sigma)_{[A]\sigma} \mid (f_{(\sigma_1 \times \ldots \times \sigma_n) \to \delta}(t_{\sigma_1}, \ldots, t_{\sigma_n}))_\delta \mid \langle t_\sigma, u_{\sigma'} \rangle_{\langle \sigma, \sigma' \rangle},$$

where $A \in \mathbb{A}$*,* $a_A \in A$*,* $x_\sigma \in V_\sigma$ *and* f *is a function symbol in* F *with arity* $(\sigma_1 \times \ldots \times \sigma_n) \to \delta$*, with* $\delta \in \Delta$*.*

When Σ is understood or irrelevant, we may write just $\mathbb{T}$ in lieu of $\mathbb{T}(\Sigma)$. The subscripts of nominal terms control sorting and we omit them when they are clear from the context or immaterial. We call $[a]t$ an *abstraction* (of a in t).

By way of examplpe, we give below a possible nominal signature for λ-terms, which is based on [41, Example 3].

Example 1. The nominal syntax of the λ-calculus is constructed using a base sort L for λ-terms, an atom sort A and the following function symbols.

- $\mathrm{var} : A \to L$: A unary function symbol for creating terms from atoms;
- $\lambda(\_) : [A]L \to L$: A unary function symbol for embedding abstractions inside terms;
- $\mathrm{app} : (L \times L) \to L$: A binary function symbol for application.

The correspondence between nominal λ-terms and those of the λ-calculus is straightforward, and will be formalized later in the paper. For example, the nominal term

$$\mathrm{app}(\lambda([a](\mathrm{app}(\mathrm{var}(a), \mathrm{var}(a)))), \lambda([a](\mathrm{app}(\mathrm{var}(a), \mathrm{var}(a)))))$$

represents the λ-term

$$\lambda a.(a\ a)\ \lambda a.(a\ a).$$

In what follows, for the sake of readability and to keep the notation for terms close to that of the λ-calculus, we apply an implicit coercion and write a in lieu of $\mathrm{var}(a)$ when atoms are used in positions where an expression of sort term is expected. Moreover, we will simply write $(M\ N)$ for $\mathrm{app}(M, N)$.

We now clarify the role of atoms and variables in Nominal SOS. As in the ordinary theory of SOS, we treat variables x, y, z, ..., as meta-variables, see [3] and [38], that range over the terms of the language, according to their sort.

On the other hand, atoms are named elements that a user can employ in programs. We call them atoms for historical reasons and, as stated in [53] with efficacy,

> [...] partly to indicate that the internal structure of such names is irrelevant to us: all we care about is their identity (i.e. whether or not one atom is the same as another) and that the supply of atoms is inexhaustible.

These named elements may serve various purposes. The most typical and also prominent example of usage of atoms is that, like formal parameters in procedures or function definitions, they represent placeholders for terms *yet to come*, in a parameter passing fashion. This is the case, for instance, in the λ-calculus [6], where atoms are employed in order to model the variables of the object language that is being formalized.

The following definitions will be useful in the remainder of the paper.

Definition 4 *Let t be a nominal term.*

- *$\mathcal{V}(t)$ denotes the set of variables that occur in t. For example, $\mathcal{V}(\lambda([a]x)) = \{x\}$.*
- *$\mathcal{A}(t)$ stands for the set of atoms that occur in t. For example, $\mathcal{A}(\lambda([a](a\ b))) = \{a, b\}$.*
- *$ba(t)$ is the set of atoms a for which there exists a subterm $[a]t'$ in t, i.e., the set of abstracted atoms in t. For example, $ba(\lambda([a](a\ b))) = \{a\}$.*
- *$fa(t)$ is the set of atoms a in $\mathcal{A}(t)$ that have an occurrence in t that is not within the scope of an abstraction $[a]t'$, for some term t'. We call $fa(t)$ the set of* free *atoms of t. For example, $fa(\lambda([a](a\ b))) = \{b\}$ and $fa(\lambda([a]a)\ a) = \{a\}$.*
- *We say that an atom a is* fresh *in t whenever $a \notin fa(t)$. We also say that a term t is* binding-closed[1] *if $fa(t) = \emptyset$, i.e., the term t does not contain free atoms.*

[1] Binding-closed terms corresponds to those that in literature are usually called *closed*. For instance, in the context of the λ-calculus, the λ-term $\lambda a.\lambda b.(a\ b)$ is closed, as it does not contain free variables, see [2, 6]. We adopt a different terminology in order to avoid confusion with the standard concept of closed term in the setting of SOS, i.e., a term that contains no variables.

These sets will play a role in some proofs to follow. Their formal definitions are thus provided in Appendix A for reference purposes.

We say that a nominal term is *closed* if it contains no variables. It is called *open* otherwise. For example, a and $[a]f(b)$ are closed terms, but x and $[a]y$ are open terms. Note that neither a nor $[a]f(b)$ is binding-closed.

The set of closed terms in $\mathbb{T}(\Sigma)$ is denoted by $\mathbb{C}(\Sigma)$ and, again, we may write just $\mathbb{C}$ when Σ is understood or irrelevant. The sets of binding-closed terms in $\mathbb{T}(\Sigma)$ and those in $\mathbb{C}(\Sigma)$ are denoted by $\mathbb{T}(\Sigma)^0$ and $\mathbb{C}(\Sigma)^0$, respectively[2]. A *substitution* ρ over the signature Σ is a function of type $V \to \mathbb{T}(\Sigma)$[3]. We assume that substitutions are *sort-respecting*, i.e., that $\rho(x)$ and x have the same sort for each $x \in V$. We extend the domain of substitutions to terms homomorphically and write $t\rho$ for the result of applying the substitution ρ to the term t. Note, that this notion of substitution is a simple operation of textual replacement and that atoms may be captured after substitution. For example, if $t = \lambda([a]x)$ and $\rho(x) = \mathrm{var}(a)$ then $t\rho = \lambda([a]\mathrm{var}(a))$. See, for instance, the introduction of [53] for a discussion of the differences between textual substitution and the capture-avoiding one we will define in what follows.

If the range of a substitution is included in $\mathbb{C}(\Sigma)$, we say that it is a *closed substitution*. That substitutions respect the sort of variables is a necessary requirement in the framework of Nominal SOS. Consider for instance the atom sort A, a base sort L and a binary function symbol f of arity $(A \times A) \to L$, i.e., the operator f accepts two atoms as arguments. A sort-respecting substitution ρ guarantees that in the term $f(x, y)\rho$, the variables x and y are mapped to atoms in A as expected, and not to some other type of terms.

3 Nominal SOS

In languages with atoms and binding operators, the proper handling of key notions such as capture-avoiding substitutions and renaming of bound atoms requires a careful treatment of free atoms. For example, in the λ-calculus one can substitute a term N for an atom b in the expression $\lambda a.\, b$ only when a is fresh (that is, is not free) in N. A first-class treatment of atom freshness is therefore an important ingredient in a nominal framework for defining operational semantics for calculi with binders—see, e.g., nominal rewriting as presented in [12].

Suppose that a is an atom and t is a term of some sort. We call a formula $a\#t$ a *freshness assertion*; intuitively, it states that the atom a is fresh in t. In what follows we give a proof system for deriving freshness assertions. However, before embarking on the formal definition, we clarify the notion of freshness by means of the following examples, which use the nominal syntax for λ-terms defined in Example 1.

[2] The notation adopted for the set of binding-closed terms follows standard practice. For instance the set of λ-terms is typically denoted by Λ and the set of closed λ-terms is typically denoted by Λ^0, see, for instance, [2, 6].

[3] We use the symbol ρ for substitutions in place of the standard symbol σ in order to avoid the confusion that would otherwise arise with sorts.

- The atom a is fresh in $\lambda([b]c)$, as it does not appear in it. Thus the assertion $a\#\lambda([b]c)$ should be derivable.
- The atom a is *not* fresh in $\lambda([b]a)$, as it appears *free* in that term. Thus, the assertion $a\#\lambda([b]a)$ should *not* be derivable.
- The atom a *is* fresh in $\lambda([b]\lambda([a]c))$. In fact, the actual bound name in the abstraction $[a]c$ is considered immaterial and so it is hidden to an external observer. Thus, the assertion $a\#\lambda([b]\lambda([a]c))$ should be derivable, and so should $b\#\lambda([b]\lambda([a]c))$.

We now proceed to formalize the derivation of freshness assertions by suitably adapting the rules in [53, Figure 2].

Definition 5 (Freshness derivation rules) *Let Σ be a nominal signature, and let the atom a and the term t be over the signature Σ. We write $\vdash a\#t$ when $a\#t$ may be derived using the following rules, where a and b are distinct atoms.*

$$\frac{}{a\#\langle\rangle} \qquad \frac{}{a\#b} \qquad \frac{a\#t_1, \ldots, a\#t_n}{a\#f(t_1, \ldots, t_n)} \qquad \frac{}{a\#[a]t} \qquad \frac{a\#t}{a\#[b]t} \qquad \frac{a\#t_1,\ a\#t_2}{a\#\langle t_1, t_2\rangle}$$

As a matter of notation, we often simply write $a\#t$ for $\vdash a\#t$. The following theorem, which can be shown using a routine inductive argument, states the correctness of the proof system for freshness assertions given above with respect to the definition of freshness given in Definition 4.

Theorem 1 (Correctness of Freshness Derivations). *Let Σ be a nominal signature and let the atom a and the term t be over the signature Σ. Then a is fresh in t if, and only if, $a\#t$.*

We are now ready to define the notion of *nominal transition system specification* whose rules employ the freshness assertions defined above.

Definition 6 (Transition relations and formulae) *Let R be a set of transition relation symbols. To each $r \in R$ we associate a* transition relation arity*, which is a sort of the form $\sigma \times \sigma_l \times \sigma'$. We call σ the 'sort of the* source *of the transition', σ' the 'sort of the* target *of the transition', and σ_l the 'sort of the* label *of the transition'.*

For a relation $r \in R$ with arity $\sigma \times \sigma_l \times \sigma'$, a positive transition formula *is written $t \xrightarrow{l}_r t'$, where t is a possibly open term of sort σ (we call it the* source *term), l is a possibly open term of sort σ_l (we call it the* label*), and t' is a possibly open term of sort σ' (we call it the* target *term).*

For the same relation r, we write $t \not\xrightarrow{l}_r$ for a negative transition formula*, where t is of sort σ and l is of sort σ_l.*

A transition formula *is a positive or negative transition formula.*

For a relation $r \in R$ with arity $\sigma \times \sigma_l \times \sigma'$, if σ_l is the unit sort $\mathbf{1}$ then we say that r *has no label*. If σ' is the unit sort, then r is a predicate symbol. We may silently drop σ_l (and σ') if they are the unit sort.

Definition 7 (Derivation rule) *A* derivation rule *is of the form*

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} t'_i \mid i \in I\} \cup \{t_j \not\xrightarrow{l_j}_{r_j} \mid j \in J\} \cup \{a_k \# t_k \mid k \in K\}}{t \xrightarrow{l}_r t'}$$

where

- I, J *and* K *are* indexing sets,
- $\{t_i \xrightarrow{l_i}_{r_i} t'_i \mid i \in I\}$ *is a set of positive transition formulae, called the* positive premises *of the rule,*
- $\{t_j \not\xrightarrow{l_j}_{r_j} \mid j \in J\}$ *is a set of negative transition formulae, called the* negative premises *of the rule,*
- $\{a_k \# t_k \mid k \in K\}$ *is a set of freshness assertions, called the* freshness premises *of the rule, and*
- $t \xrightarrow{l}_r t'$ *is a positive transition formula, which we call the* conclusion *of the rule.*

We call t, l, *and* t' *the* source, *the* label *and the* target *of the rule, respectively.*

We call a derivation rule an axiom *if* I, J *and* K *are empty. A derivation rule is* positive *when the index set* J *is empty.*

Substitutions are also extended to formulae, sets of formulae and rules in the expected way. For a derivation rule d and a substitution ρ, the rule $d\rho$ is called a *substitution instance* of d.

Definition 8 (Nominal Transition System Specification) *A* nominal transition system specification *(*NTSS*) is a triple* (Σ, R, D) *consisting of:*

1. *A nominal signature* Σ*;*
2. *A set of* (transition) relation symbols R*; and*
3. *A set of derivation rules* D*.*

An NTSS is *positive* when all its deduction rules are positive. Positive NTSS's are much easier to deal with than general ones and come with a natural notion of semantics, i.e., the set of provable transitions.

Given closed terms t and t', and a label l, the intended reading of $t \xrightarrow{l}_r t'$ is: t can make an r-transition with label l to t'. We write $t \not\xrightarrow{l}_r$ to mean that there is no term t' such that $t \xrightarrow{l}_r t'$. A positive NTSS gives these intuitions formal meaning using a notion of 'derivable transition', which we now define.

Definition 9 *Let* T *be an NTSS. The* derivable transitions *of* T *are inductively defined as follows. Suppose that*

$$\frac{\{t_i \xrightarrow{l_i}_{r_i} t'_i \mid i \in I\} \cup \{a_k \# t_k \mid k \in K\}}{t \xrightarrow{l}_r t'}$$

is a rule in T*, and suppose* ρ *is a closed substitution over the signature of* T*. If*

– $t_i\rho \xrightarrow{l_i\rho}_{r_i} t'_i\rho$ *is derivable, for every* $i \in I$*, and*
– $a_k \# t_k\rho$ *is derivable (using the rules in Definition 5), for every* $k \in K$*,*

then $t\rho \xrightarrow{l\rho}_r t'\rho$ *is derivable.*

This means that the set of derivable transitions is the least collection of transitions that is 'closed under the application of the rules.'

3.1 Semantics of NTSSs

All the NTSSs we consider in this paper are positive, and Definition 9 suffices to give their semantics.

To give a semantics to NTSSs in general, one has to define a meaning for negative transitions, i.e., when a negative transition is 'provable'. This has been a source of complications in the theory of SOS and several proposals for such a notion exist [54]. The most widely accepted notion of semantics for transition system specifications involving negative transitions is that of least three-valued stable model. To define this concept, we need two auxiliary definitions, namely provable transition rules and consistency, which are given below.

Definition 10 (Provable transition rules) *A closed deduction rule is called a* transition rule *when it is of the form* $\frac{N}{\phi}$ *with* N *a set of* negative formulae*. An NTSS* T proves $\frac{N}{\phi}$*, denoted by* $T \vdash \frac{N}{\phi}$*, when there is a well-founded upwardly branching tree with closed formulae as nodes and of which*

– *the root is labelled by* ϕ*;*
– *if a node is labelled by* ψ *and the labels of the nodes directly above it form the set* K *then:*
  - ψ *is a negative formula and* $\psi \in N$*, or*
  - ψ *is a positive formula and* $\frac{K}{\psi}$ *is a substitution instance of a deduction rule in* T*.*

Definition 11 (Contradiction and consistency) *The closed transition formula* $t \xrightarrow{l}_r t'$ *is said to* contradict $t \not\xrightarrow{l}_r$*, and vice versa. For two sets* Φ *and* Ψ *of closed formulae,* Φ contradicts Ψ *when there is some* $\phi \in \Phi$ *that contradicts a* $\psi \in \Psi$*. We write* $\Phi \models \Psi$*, and say that* Φ entails Ψ*, when* Φ *does not contradict* Ψ *and each positive transition formula in* Ψ *is contained in* Φ*.*

Remark 1. Note that, when Ψ is a collection of negative transition formulae, $\Phi \models \Psi$ holds exactly when Φ does not contradict Ψ.

We now have all the necessary ingredients to define the semantics of NTSSs in terms of three-valued stable models.

Definition 12 (Three-valued stable model) *A pair* (C, U) *of disjoint sets of positive closed transition formulae is called a* three-valued stable model *for an NTSS* T *when the following conditions hold for each* ϕ*:*

– $\phi \in C$ *if, and only if, there is a set* N *of negative formulae such that* $T \vdash \frac{N}{\phi}$ *and* $C \cup U \vDash N$*, and*
– $\phi \in C \cup U$ *if, and only if, there is a set* N *of negative formulae such that* $T \vdash \frac{N}{\phi}$ *and* $C \vDash N$*.*

C *stands for* Certainly *and* U *for* Unknown*; the third value is determined by the formulae not in* $C \cup U$*. The* least *three-valued stable model is a three-valued stable model that is the least one with respect to the ordering on pairs of sets of formulae defined as* $(C, U) \leq (C', U')$ *iff* $C \subseteq C'$ *and* $U' \subseteq U$*.*

In the literature [44, 54] (in the setting without names and binders), it has been shown that every TSS admits a least three-valued stable model with respect to the information theoretic ordering. This result easily extends also to our setting for NTSSs.

We say that an NTSS is *complete* when for its least three-valued stable model it holds that $U = \emptyset$. If an NTSS is complete, we write $p \xrightarrow{l}_r p'$ in lieu of $(p \xrightarrow{l}_r p') \in C$. All the NTSSs considered in this paper are positive and therefore complete.

4 Substitution and α-conversion

Substitution and α-equivalence play a key role in the definition of the semantics of calculi with binders. We will now show how those notions can be accommodated within the framework of Nominal SOS.

Atoms inequality First of all, notice that it is possible to employ freshness assertions in order to check for inequality between two atoms. As a matter of fact, the assertion $a\#b$ holds exactly when a and b are different atoms. In the remainder of this paper, some rules need to check inequality of atoms. In order to increase the readability of rules, and let the reader understand clearly the meaning behind some premises, we shall write sometimes a premise $a \neq t$ to mean $a\#t$, with t an atom or a variable of atom sort. The checking of atoms equality is never employed in the remainder of this paper. As future extensions of the framework of Nominal SOS, we are planning to allow also for negative freshness premises and this type of premises can be used to express equality of atoms, as an atom a is not fresh in an atom b only when a and b are the same atom.

4.1 Substitution transitions

Substitution is a natural notion required in the semantics of most nominal calculi. The well-known notion of substitution for these calculi is subtly different from the naive textual substitution we have used in giving semantics to NTSSs in that it should avoid 'capture of atoms'. For example, replacing b with a in $[a]b$, if done naively, results in $[a]a$, which captures the free name a by the abstraction surrounding it. Hence, a substitution is allowed only as long as the variable that

is abstracted by the binder is fresh in the substituting term. In Nominal SOS, we can model this freshness requirement within our framework (instead of in the meta-language as in most other approaches, such as the one presented in [9]) by means of freshness assertions, which are native to the nominal framework.

The substitution discussed in the previous paragraph is what we will call an *atom-for-atom* substitution, since it replaces atoms with atoms. Such substitutions underlie the definition of α-conversion and α-equivalence (Section 4.2) and are used in calculi such as the π-calculus, whose formulation within the Nominal SOS franework we will discuss in Section 5.2. The effect of applying an atom-for-atom substitution to terms over a nominal signature is described by means of a transition relation. For a given nominal signature, the deduction rules for atom-for-atom substitution transitions can be generated automatically following the procedure that we will present shortly.

Capture avoiding substitutions that replace atoms with terms are typically employed by higher-order calculi, such as the λ-calculus, CHOCS [52] and the Higher-order π-calculus [45], just to mention a few. The effect of applying such substituions to terms can be axiomatized using calculus-specific deduction rules, as we exemplify in Section 5.1 in the setting of the λ-calculus.

Atom-for-atom substitution is used in calculi such as the π-calculus [48, 36] and its variants. In Nominal SOS the effect of applying such substitutions on terms is described by means of the deduction rules in Figure 1. Those rules generate transitions of the form $t_1 \xrightarrow{a \overset{A}{\mapsto} b} t_2$ for some atoms a and b of the same sort and terms t_1 and t_2. More specifically, given an atom sort A such that a and b are of sort A, and a sort σ, the transition relation $\xrightarrow{a \overset{A}{\mapsto} b}$ has sort $\sigma \times (A \times A) \times \sigma$; even though its sort is $A \times A$, we write $a \overset{A}{\mapsto} b$ in lieu of $\langle a, b \rangle$ for the label of a substituion transition to emphasize the fact that such a transition describes the effect of replacing atom a by atom b in a term.

$$a \xrightarrow{a \overset{A}{\mapsto} z} z \ (\mathrm{a1_{As}}) \qquad \frac{a \neq x}{a \xrightarrow{x \overset{A}{\mapsto} z} a} \ (\mathrm{a2_{As}}) \qquad \frac{x \xrightarrow{y \overset{A}{\mapsto} z} x' \quad a \neq z \quad a \neq y}{[a]x \xrightarrow{y \overset{A}{\mapsto} z} [a]x'} \ (\mathrm{abs1_{As}})$$

$$[a]x \xrightarrow{a \overset{A}{\mapsto} z} [a]x \ (\mathrm{abs2_{As}}) \qquad \frac{\left\{ x_i \xrightarrow{y \overset{A}{\mapsto} z} x_i' \mid 1 \leq i \leq n \right\}}{f(x_1, x_2, \ldots, x_n) \xrightarrow{y \overset{A}{\mapsto} z} f(x_1', x_2', \ldots, x_n')} \ (\mathrm{f_{As}})$$

$$\langle\rangle \xrightarrow{y \overset{A}{\mapsto} z} \langle\rangle \ (\mathrm{u_{As}}) \qquad \frac{x_1 \xrightarrow{y \overset{A}{\mapsto} z} x_1', \ x_2 \xrightarrow{y \overset{A}{\mapsto} z} x_2'}{\langle x_1, x_2 \rangle \xrightarrow{y \overset{A}{\mapsto} z} \langle x_1', x_2' \rangle} \ (\mathrm{p_{As}})$$

Fig. 1. Deduction rules for atom-for-atom substitution transitions

The sort of the variables used in the rules in Figure 1 can be inferred by their usage. For instance, the variable z that occurs in the rule ($\mathrm{a2}_{\mathrm{As}}$) is of atom sort. Recall also that a premise $a \neq x$, where x is a variable ranging over an atom sort, means $a\#x$.

Lemma 1.

1. *Atom-for-atom substitution transitions are sort respecting, that is, if* $t \xrightarrow{a \overset{A}{\mapsto} b} u$ *then* t *and* u *are terms of the same sort.*
2. *If* $t \xrightarrow{b \overset{A}{\mapsto} a} u$ *and* $b\#t$ *then* $t \equiv u$.
3. *If* $t \xrightarrow{a \overset{A}{\mapsto} b} u$ *and* $b\#t$ *then* $u \xrightarrow{b \overset{A}{\mapsto} a} t$ *and* $a\#u$.

Proof. Statements 1 and 3 can be shown by induction on the proof of the transition $t \xrightarrow{a \overset{A}{\mapsto} b} u$. The proof of statement 3 uses statement 2, which can be shown by induction on the structure of t. ⊠

It is important not to confuse the textual substitutions ρ defined in Section 2 and the one defined above. The former is on the meta-level of the semantics of Nominal SOS and acts on variables. It is a semantic meta-operation that is used in order to instanciate the rules of Nominal SOS and prove transitions. Moreover, this notion allows for capture of atoms. For instance, we can apply the axiom ($\mathrm{abs2}_{\mathrm{As}}$) with a substitution ρ that maps x to a and z to b, for some atoms a and b, in order to prove the transition $[a]a \xrightarrow{a \overset{A}{\mapsto} b} [a]a$.

Instead, the purpose of atom-for-atom substitutions and of the term-for-atom substitutions we will define in Section 5.1 is replacing atoms by suitable terms, thus modelling the fact that atoms are variables of the object language and represent placeholders for other terms.

Recall also that the textual substitutions ρ that are employed in the definition of the semantics of an NTSS must respect the sort of variables. This means, for instance, that the rules above cannot be applied to prove term-for-atom substitution transitions. For example, given a complex (non-atom) term t, the transition $a \xrightarrow{a \overset{A}{\mapsto} t} t$ is not provable using rule ($a1_{\mathrm{As}}$).

Atom-for-atom substitution transitions are deterministic.

Lemma 2 (Determinism of substitution transitions). *Let* T *be an NTSS containing the rules for atom-for-atom substitution as defined in Figure 1. For all closed terms* t, t', t'', *and for all atoms* a, b, *it holds that if* $t \xrightarrow{a \overset{A}{\mapsto} b} t'$ *and* $t \xrightarrow{a \overset{A}{\mapsto} b} t''$ *then* $t' = t''$.

The above lemma can be proved easily by structural induction. To see this, it suffices only to notice that rules ($\mathrm{a1}_{\mathrm{As}}$) and ($\mathrm{a2}_{\mathrm{As}}$) cannot be applied simultaneously, and also ($\mathrm{abs1}_{\mathrm{As}}$) and ($\mathrm{abs2}_{\mathrm{As}}$) cannot be applied simultaneously.

The reader may be more familiar with the syntactic substitution operation, defined below, where t and the t_is are closed terms and a, b, c are atoms with a

and b are distinct.

$$\begin{aligned}
\langle\rangle[b/a] &= \langle\rangle \\
a[b/a] &= b \\
a[c/b] &= a \\
([a]t)[b/a] &= [a]t \\
([a]t)[c/b] &= [a](t[c/b]) \quad \text{if } a \neq c \\
f(t_1, t_2, \ldots, t_n)[b/a] &= f(t_1[b/a], t_2[b/a], \ldots, t_n[b/a]) \\
\langle t_1, t_2\rangle[b/a] &= \langle t_1[b/a], t_2[b/a]\rangle
\end{aligned}$$

The following theorem states that the two notions (substitution transitions and syntactic substitutions) correspond.

Theorem 2 (Correctness of Substitution Transitions). *Let T be an NTSS. Let t and t' be closed terms, and a and b be atoms. Then $t \xrightarrow{a \overset{A}{\longmapsto} b} t'$ if, and only if, $t' = t[b/a]$.*

We give the proof of Theorem 2 in Appendix D.

4.2 α-conversion Transitions

The notion of α-congruence is a natural equivalence relation over terms guaranteeing that the name chosen in binders is not important and can be indeed replaced by any other name (while avoiding capture). Unfortunately, not all names can be picked when performing this change. To exemplify this fact, we can consider again the λ-calculus. The term $\lambda a.(a\ b)$ is α-equivalent to $\lambda c.(c\ b)$ (provided that $a \neq b$ and $b \neq c$). The atom c is indeed a suitable atom for 'α-conversion'. However, the atom b is not suitable, because one does not want to α-convert the term $\lambda a.(a\ b)$ into $\lambda b.(b\ b)$ as this leads to the capture of a free atom. Again, thanks to freshness assertions, we can accommodate α-conversion in our framework as an ordinary transition relation. Given a nominal signature Σ, the following deduction rules define the relation $\approx_\alpha$ over closed terms over Σ. (Strictly speaking, for each sort σ, there is a relation $\approx_\alpha^\sigma$ with sort $\sigma \times \mathbf{1} \times \sigma$, and these transition relations are defined using the rules below. In what follows we omit the sort information and the label from the relation for the sake of redability.) For all atoms a and b and function symbols f, we have the rules in Figure 2. Note that α-conversion transitions rely on those for the atom-for-atom substitution (see rule (abs1_α)). Throughout the paper, whenever we say that the above rules for α-conversion transitions are present in an NTSS, this implies that also the rules for atom-for-atom substitutions are present.

Lemma 3. *The transition relation $\approx_\alpha$ is sort respecting, that is, if $t_1 \approx_\alpha t_2$ then t_1 and t_2 are terms of the same sort. Moreover, $\approx_\alpha$ is a congruence relation over terms.*

Proof. The first claim can be shown by induction on the proof of the transition $t_1 \approx_\alpha t_2$, using Lemma 1(1) in case the last rule used in the proof is (abs1_α).

$$a \approx_\alpha a \ (\mathrm{id}_\alpha) \qquad \langle\rangle \approx_\alpha \langle\rangle \ (\mathrm{idU}_\alpha) \qquad \frac{x \approx_\alpha y \quad y \approx_\alpha z}{x \approx_\alpha z} \ (\mathrm{Trans}_\alpha)$$

$$\frac{x \xrightarrow{a \overset{A}{\mapsto} b} y \qquad b\#x}{[a]x \approx_\alpha [b]y} \ (\mathrm{abs1}_\alpha) \qquad \frac{x \approx_\alpha y}{[a]x \approx_\alpha [a]y} \ (\mathrm{abs2}_\alpha)$$

$$\frac{\{x_i \approx_\alpha x'_i \mid 1 \leq i \leq n\}}{f(x_1, x_2, \ldots, x_n) \approx_\alpha f(x'_1, x'_2, \ldots, x'_n)} \ (\mathrm{f}_\alpha) \qquad \frac{x_1 \approx_\alpha x'_1, \ x_2 \approx_\alpha x'_2}{\langle x_1, x_2\rangle \approx_\alpha \langle x'_1, x'_2\rangle} \ (\mathrm{p}_\alpha)$$

Fig. 2. Deduction rules axiomatizing $\approx_\alpha$

Reflexivity of $\approx_\alpha$ follows by induction on the structure of closed terms and transitivity is immediate by rule (Trans_α).

To prove that $\approx_\alpha$ is symmetric, one shows that if $t \approx_\alpha u$ then $u \approx_\alpha t$ by induction on the proof of the transition $t \approx_\alpha u$. The only non-trivial case in the proof is when $t \approx_\alpha u$ has been shown using rule $(\mathrm{abs1}_\alpha)$, which is the one asymmetric deduction rule for $\approx_\alpha$. In that case, we have that

- $t \equiv [a]t'$ for some t',
- $u \equiv [b]u'$ for some u',
- $t' \xrightarrow{a \overset{A}{\mapsto} b} u'$ and
- $b\#t'$.

Since $t' \xrightarrow{a \overset{A}{\mapsto} b} u'$ and $b\#t'$, Lemma 1(3) yields $u' \xrightarrow{b \overset{A}{\mapsto} a} t'$ and $a\#u'$. Therefore, using rule $(\mathrm{abs1}_\alpha)$, we may infer that $u \equiv [b]u' \approx_\alpha [a]t' \equiv t$.

The substitutivity of $\approx_\alpha$ is an immediate consequence of rules $(\mathrm{abs2}_\alpha)$, (f_α) and (p_α). ⊠

Example 2. Rule (Trans_α) is necessary to ensure trasitivity of $\approx_\alpha$. To see this, consider the terms

$$\begin{aligned} t &= [a][c]f(a, c), \\ u &= [b][c]f(b, c) \qquad \text{and} \\ v &= [b][d]f(b, d). \end{aligned}$$

Using rule $(\mathrm{abs1}_\alpha)$ one can show that $t \approx_\alpha u$ and $u \approx_\alpha v$. On the other hand, without rule (Trans_α), it is impossible to derive $t \approx_\alpha v$.

The reader will be familiar with the syntactic version of α-conversion, defined below.

Definition 13 (α-conversion over nominal terms) *Let T be an NTSS. The relation $=_\alpha$ is the least congruence over nominal terms over the signature of T, such that, for each closed term t and for each atom b, if b is fresh in t then $[a]t =_\alpha [b](t[b/a])$. (Recall that 'b is fresh in t' means that $b \notin fa(t)$.)*

The relations $\approx_\alpha$ and $=_\alpha$ agree, as stated in the following theorem.

Theorem 3 (Correctness of α-conversion transitions). *Let T be an NTSS. For all closed terms t and u over the signature of T, it holds that $t \approx_\alpha u$ if, and only if, $t =_\alpha u$.*

Proof. Let T be an NTSS. We show that. for all closed terms t and u over the signature of T, $t \approx_\alpha u$ if, and only if, $t =_\alpha u$.

The implication from left to right can be proved by induction on the proof of the transition $t \approx_\alpha u$. Since the details are not hard, we limit ourselves to presenting the most interesting case in the proof, namely, the one when $t \approx_\alpha u$ has been derived using rule (abs1$_\alpha$). In this case, we have that

- $t \equiv [a]t'$ for some t',
- $u \equiv [b]u'$ for some u',
- $t' \xrightarrow{a \overset{A}{\mapsto} b} u'$ and
- $b\#t'$.

Since $t' \xrightarrow{a \overset{A}{\mapsto} b} u'$ and $b\#t'$, we have that $u' = t'[b/a]$ (Theorem 2) and b is fresh in t' (Theorem 1). So, by Definition 13, $t \equiv [a]t' =_\alpha [b](t'[b/a]) \equiv u$ and we are done.

The implication from right to left can be shown by induction on the proof of $t =_\alpha u$, using the fact that $\approx_\alpha$ is a congruence relation by Lemma 3. In the case that $t =_\alpha u$ because $t \equiv [a]t'$ for some t', $u \equiv [b](t[b/a])$ and b is fresh in t', one proceeds along the lines given above. ⊠

Calculi with binders usually consider a term as a representative of the equivalence class of all the terms that are α-convertible to it. In Nominal SOS, it is possible to achieve this affect by augmenting a NTSS with a deduction rule, given below.

Definition 14 (Transitions up to α-equivalence) *Let T be an NTSS and l be a label over the signature of T. The transition relation $\xrightarrow{l}$ is* up to α-equivalence *whenever the deduction rules of T contain the rules for α-conversion transitions, as defined above, the rules for atom-for-atom substitution transitions, as defined in Figure 1, and the deduction rule:*

$$\frac{x \approx_\alpha y \quad y \xrightarrow{l} w \quad w \approx_\alpha z}{x \xrightarrow{l} z} \ (l \cdot \text{upTo}\alpha).$$

Depending on the characteristics of the calculus at hand, the modeller might want to define some of the transition relations to be up to α-equivalence.

5 Examples

As we have seen so far, Nominal SOS is an extension of standard SOS with atoms, binding constructs and freshness assertions. Freshness assertions may be

used in the inference rules that define the transition relations describing the operational semantics of terms and can be proved to hold using a proof system that is uniformly defined from a given nominal signature. Notions like atom-for-atom substitution and α-equivalence are part of the framework, rather than being meta-level notions, and are defined by rules that are also automatically generated from a given nominal signature.

To our mind, these are pleasing properties of the framework. However, a key question to be addressed at this point is whether Nominal SOS is actually useful in formalizing the operational semantics of nominal languages.

In this section we provide some evidence for the expressiveness, and perhaps naturalness, of Nominal SOS by formulating in our framework two classic nominal calculi, namely the lazy λ-calculus [2] and the early π-calculus [36, 48]. Earlier nominal formulations of the λ-calculus may be found in, e.g., [17, 41].

5.1 The lazy λ-calculus

For ease of reference, we repeat here the signature for λ-terms given in Example 1. The signature Σ^{λ} of the lazy λ-calculus is constructed using a base sort L for λ-terms and an atom sort A. The signature also contains the following function symbols:

- $\mathrm{var} : A \to L$: A unary function symbol for creating terms from atoms;
- $\lambda(\_) : [A]L \to L$: A unary function symbol for embedding abstractions inside terms;
- $\mathrm{app} : (L \times L) \to L$: A binary function symbol for application.

The operational semantics for the language is given in terms of a big-step reduction transition $\to$, whose sort is $L \times \mathbf{1} \times L$, here displayed with no label to remain in line with the standard notation from [2]. Since in the λ-calculus atoms are placeholders for terms, the NTSS for the nominal formulation of that calculus also needs to implement term-for-atom substitutions. As we did for atom-for-atom substitutions, we describe the effect of replacing an atom with a term in a nominal λ-term using a transition relation, whose sort is $L \times (A \times L) \times L$. The rules that axiomatize term-for-atom substitution transitions are given in Figure 3. Those rules generate transitions of the form $M \xrightarrow{a \overset{T}{\mapsto} N} M'$ for some atom a and terms M, M' and N. (We write $a \overset{T}{\mapsto} N$ in lieu of $\langle a, N \rangle$ for the label of a substituion transition to emphasize the fact that such a transition describes the effect of replacing atom a by term N in a term.)

In the rules in Figure 3, we have used the formal definition of the signature for nominal λ-terms. However, to ease readability and to keep the notation for terms close to that of the λ-calculus, in the remainder of this paper we will write simply a instead of the term $\mathrm{var}(a)$ for λ-calculus variables and $(M\ N)$ for $\mathrm{app}(M, N)$..

The set of rules of the NTSS T_{λ} for the lazy λ-calculus contains the following two deduction rules, which define the operational semantics of our version of the

$$\mathrm{var}(a) \xrightarrow{a \overset{T}{\mapsto} z} z \ (\mathrm{a1_{Ts}}) \qquad \frac{a \neq x}{\mathrm{var}(a) \xrightarrow{x \overset{T}{\mapsto} z} \mathrm{var}(a)} \ (\mathrm{a2_{Ts}})$$

$$\frac{x \xrightarrow{y \overset{T}{\mapsto} z} x' \quad a\#z \quad a \neq y}{[a]x \xrightarrow{y \overset{T}{\mapsto} z} [a]x'} \ (\mathrm{abs1_{Ts}}) \qquad [a]x \xrightarrow{a \overset{T}{\mapsto} z} [a]x \ (\mathrm{abs2_{Ts}})$$

$$\frac{x \xrightarrow{y \overset{T}{\mapsto} z} x'}{\lambda(x) \xrightarrow{y \overset{T}{\mapsto} z} \lambda(x')} \ (\lambda_{\mathrm{Ts}}) \qquad \frac{x \xrightarrow{w \overset{T}{\mapsto} z} x' \quad y \xrightarrow{w \overset{T}{\mapsto} z} y'}{\mathrm{app}(x,y) \xrightarrow{w \overset{T}{\mapsto} z} \mathrm{app}(x',y')} \ (\mathrm{app_{Ts}})$$

Fig. 3. Deduction rules for term-for-atom substitution transitions in the λ-calculus

lazy λ-calculus, for all atoms a.

$$\frac{}{\lambda([a]x) \to \lambda([a]x)} \ (\mathrm{abs}) \qquad \frac{x_0 \to \lambda([a]y_0) \quad y_0 \xrightarrow{a \overset{T}{\mapsto} x_1} y_1 \quad y_1 \to y_2}{(x_0\ x_1) \to y_2} \ (\mathrm{app})$$

The transition relation $\to$ and the term-for-atom substitution transition relations are up to α-equivalence. Notice that, by Definition 14, this means that the set of rules of T_λ contains also the rules for α-conversion transitions and the rules for atom-for-atom substitution transitions, which are themselves set to be up to α-equivalence.

Definition 1. *We say that a term $M \in \mathbb{C}(\Sigma^\lambda)$ has a* normal form *if $M \to M'$ for some M'. In that case, M' is a normal form for M.*

For instance, the term

$$\Omega = \lambda([a](a\ a))\ \lambda([a](a\ a))$$

has no normal form, but every abstraction does. In particular, $\lambda([a]\Omega)$ has a normal form, unlike Ω.

Following standard notation, we denote by Λ be the set of λ-terms of [2, 6]. The encoding $[\![\cdot]\!]^\lambda$ is a map from Λ into terms of our nominal λ-calculus and is defined as follows.

$$\begin{aligned} [\![a]\!]^\lambda &= a \\ [\![\lambda a.M]\!]^\lambda &= \lambda([a][\![M]\!]^\lambda) \\ [\![M\ N]\!]^\lambda &= [\![M]\!]^\lambda\ [\![N]\!]^\lambda \end{aligned}$$

We present the rules for the semantics of the original lazy λ-calculus from [2] below[4].

$$\frac{}{\lambda a.x \to \lambda a.x} \ (\mathrm{absO}) \qquad \frac{x_0 \to \lambda a.y_0 \quad y_0[x_1/a] \to y_1}{(x_0\ x_1) \to y_1} \ (\mathrm{appO})$$

[4] In [2], Abramsky uses the symbol $\Downarrow$ in lieu of $\to$.

The reader should not confuse the substitution operation over λ-terms employed in the rule (appO) with the syntactic substitution over nominal terms defined in Section 4.1. We recall that λ-terms are considered up to α-equivalence; this means that, for instance, the λ-terms $\lambda a.a$ and $\lambda b.b$ are considered syntactically equal. The notions of substitution and α-equivalence over λ-calculus terms are standard and are therefore not provided in the main body of the paper. However, for completeness, they are repeated in Appendix B together with some other useful definitions.

The following theorem establishes the operational correctness of our formulation of the lazy λ-calculus with respect to Abramsky's original one.

Theorem 4 (Operational Correspondence: lazy λ-calculus).

1. *For all $M, N \in \Lambda$, if $M \to N$ then $[\![M]\!]^\lambda \to [\![N]\!]^\lambda$.*
2. *For all $M \in \Lambda$ and $N \in \mathbb{C}(\Sigma^\lambda)$, if $[\![M]\!]^\lambda \to N$ then $M \to N'$ for some $N' \in \Lambda$ such that $[\![N']\!]^\lambda = N$.*

We give the proof of Theorem 4 in Appendix E.

5.2 The early π-calculus

The signature Σ^π of our nominal formulation of the syntax of the π-calculus contains the base sorts Act and P (for action labels and processes, respectively), the atom sort C (for channels) and the following function symbols for constructing terms of sort P:

- $\mathbf{0} :\to P$ for inaction (deadlock),
- $\tau._- : P \to P$ for τ-prefix,
- $out(\_, \_, \_) : (C \times C \times P) \to P$ for output prefix,
- $in(\_, \_) : (C \times [C]P) \to P$ for input prefix,
- $\nu(\_) : [C]P \to P$ for restriction,
- $\_ \mid \_ : (P \times P) \to P$ for parallel composition,
- $\_ + \_ : (P \times P) \to P$ for nondeterministic choice, and
- $!\_ : P \to P$ for parallel replication.

In addition, the following function symbols for building terms of sort Act are in Σ^π:

- $\tau A :\to Act$ for the τ action label,
- $inA : (C \times C) \to Act$ for constructing input actions,
- $outA : (C \times C) \to Act$ for constructing output actions, and
- $boutA : (C \times [C]\mathbf{1}) \to Act$ for constructing bound output actions.

Since our aim in this section is to provide an example of usage of Nominal SOS, we prefer to stick with the exact definitions of the framework rather than making the syntax closer to that of the π-calculus by providing syntactic sugar, as we did for the lazy λ-calculus. For instance, the syntax employed for input and output prefixes differs slightly from the standard notation used in the π-calculus. In

particular, a term $out(a, b, p)$ over Σ^π corresponds to the process $\overline{a}b.p$ of the π-calculus, and $in(a, [b]p)$ corresponds to $a(b).p$. A term of the form $\nu([a]p)$ is the nominal counterpart of what would be written $\nu a.p$ in the π-calculus.

A similar choice is adopted for the labels. We use τA for the nominal counterpart of the τ action in the π-calculus. The term $inA(a, b)$ stands for an input of channel b on channel a, which is denoted by ab in the π-calculus, whereas $outA(a, b)$ is the nominal counterpart of an output of channel b on channel a, which is denoted by $\overline{a}b$ in the π-calculus. The term $boutA(a, [b]\langle\rangle)$ represents the output of the *bound* channel name b on channel a, which is written $\overline{a}(b)$ in the π-calculus. (In what follows, we will abbreviate $boutA(a, [b]\langle\rangle)$ to $boutA(a, [b])$.)

The semantics of the early π-calculus is specified in Nominal SOS by means of the rules in Figure 4, where we use ℓ to range over labels (closed terms of base sort Act), and a, b and c to range over atoms (channels). Moreover, for each label ℓ, the transition relation $\xrightarrow{\ell}$ is up to α-equivalence. As in the previous case, by Definition 14 this means that the set of rules over Σ^π contains the rules for α-conversion transitions and the rules for atom-for-atom substitution transitions. The latter are needed both in the context of α-conversion and in the rules that specifically define our formulation of the π-calculus; see rule (in). We set the atom-for-atom substitution transition relations to be up to α-equivalence, too.

Note that the complicated side conditions of the ordinary formulation of π-calculus are replaced here by simpler freshness conditions; see, for instance, rules (parRes1) and (close1). By way of example, consider the rule of the original π-calculus that describes the interleaving semantics of the parallel composition operator. (The complete set of rules is given in Figure 5 later on in this section.) That rule is

$$bn(\ell) \cap fn(x_2) = \emptyset \;\frac{x_1 \xrightarrow{\ell} y_1}{x_1 \,||\, x_2 \xrightarrow{\ell} y_1 \,||\, x_2}\;(\text{parO1})$$

where $bn(\ell)$ denotes the set of bound names that occur in the label ℓ and $fn(p)$ denotes the set of names that have a free occurrence in the term p; see [48] and Appendix C for the formal definitions.

The side condition for this rule is 'external' to the semantic specification. In implementations, checking whether this side condition holds is handled by a run-time check, after a substitution is applied in order to use the rule (parO1) with concrete terms. On the theoretical side another approach is to populate the SOS semantics with copies of the same rule for each combination of label, substituted for ℓ, and closed term, substituted for x_2, that satisfies the side condition. This approach complicates the theory. As a result, proofs in the context of the π-calculus are usually not uniform in their development, having the necessity to switch, from time to time, from handling technicalities within the semantics of the calculus to technicalities concerning the external level of the side conditions.

In our formulation of the π-calculus, the behaviour of the rule (parO1) is modelled by considering two cases. In the first case we define the interleaving semantics of the parallel composition for all the labels that do not need the considered special checks. These are all the labels with the exception of those

$$\frac{}{\tau.x \xrightarrow{\tau A} x}\ (\tau) \qquad \frac{}{out(a,b,x) \xrightarrow{outA(a,b)} x}\ (\text{out}) \qquad \frac{x \xrightarrow{b\mapsto^{A}c} y}{in(a,[b]x) \xrightarrow{inA(a,c)} y}\ (\text{in})$$

$$\frac{x_1 \xrightarrow{\ell} y_1}{x_1 + x_2 \xrightarrow{\ell} y_1}\ (\text{sum1}) \qquad \ell \notin \{boutA(a,[b]) \mid a,b \in C\}\ \frac{x_1 \xrightarrow{\ell} y_1}{x_1 \,||\, x_2 \xrightarrow{\ell} y_1 \,||\, x_2}\ (\text{par1})$$

$$\frac{x_2 \xrightarrow{\ell} y_2}{x_1 + x_2 \xrightarrow{\ell} y_2}\ (\text{sum2}) \qquad \ell \notin \{boutA(a,[b]) \mid a,b \in C\}\ \frac{x_2 \xrightarrow{\ell} y_2}{x_1 \,||\, x_2 \xrightarrow{\ell} x_1 \,||\, y_2}\ (\text{par2})$$

$$\frac{x_1 \xrightarrow{boutA(a,[b])} y_1 \quad b\#x_2}{x_1 \,||\, x_2 \xrightarrow{boutA(a,[b])} y_1 \,||\, x_2}\ (\text{parRes1}) \qquad \frac{x_2 \xrightarrow{boutA(a,[b])} y_2 \quad b\#x_1}{x_1 \,||\, x_2 \xrightarrow{boutA(a,[b])} x_1 \,||\, y_2}\ (\text{parRes2})$$

$$\frac{x_1 \xrightarrow{outA(a,b)} y_1 \quad x_2 \xrightarrow{inA(a,b)} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau A} y_1 \,||\, y_2}\ (\text{com1}) \qquad \frac{x_1 \xrightarrow{inA(a,b)} y_1 \quad x_2 \xrightarrow{outA(a,b)} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau A} y_1 \,||\, y_2}\ (\text{com2})$$

$$\frac{x_1 \xrightarrow{boutA(a,[b])} y_1 \quad x_2 \xrightarrow{inA(a,b)} y_2 \quad b\#x_2}{x_1 \,||\, x_2 \xrightarrow{\tau A} \nu([b](y_1 \,||\, y_2))}\ (\text{close1})$$

$$\frac{x_1 \xrightarrow{inA(a,b)} y_1 \quad x_2 \xrightarrow{boutA(a,[b])} y_2 \quad b\#x_1}{x_1 \,||\, x_2 \xrightarrow{\tau A} \nu([b](y_1 \,||\, y_2))}\ (\text{close2})$$

$$\frac{x \xrightarrow{\ell} y}{!x \xrightarrow{\ell} y \,||\, !x}\ (\text{repl}) \qquad \frac{x \xrightarrow{outA(a,b)} y_1 \quad x \xrightarrow{inA(a,b)} y_2}{!x \xrightarrow{\tau A} (y_1 \,||\, y_2) \,||\, !x}\ (\text{repl} - \text{com})$$

$$\frac{x \xrightarrow{boutA(a,[b])} y_1 \quad x \xrightarrow{inA(a,b)} y_2 \quad b\#x}{!x \xrightarrow{\tau A} \nu([b](y_1 \,||\, y_2)) \,||\, !x}\ (\text{repl} - \text{close})$$

$$\frac{x \xrightarrow{outA(c,a)} y \quad a \neq c}{\nu([a]x) \xrightarrow{boutA(c,[a])} y}\ (\text{open}) \qquad c \notin ba(\ell)\ \ \frac{x \xrightarrow{\ell} y \quad c\#\ell}{\nu([c]x) \xrightarrow{\ell} \nu([c]y)}\ (\text{res})$$

Fig. 4. Rules for the nominal formulation of the early π-calculus

representing bound outputs and, for each such label, we have an instance of rule (par1). In the second case, we manage the labels that require the extra checks, namely the bound outputs. Thanks to the freshness premises, we are able to model the checks within the framework; see rule (parRes1).

As another example, consider the following rule from the early π-calculus (rule (resO) in Figure 5), where $names(\ell)$ is the set of all channel names occurring in the label ℓ:

$$c \notin names(\ell) \quad \frac{x \xrightarrow{\ell} y}{\nu c.x \xrightarrow{\ell} \nu c.x}$$

The nominal counterpart of this rule is (res) in Figure 4. The side condition of the above rule is implemented by means of a freshness premise that ensures that the rule can only be applies when the channel name c is fresh (that is, does not occur free) in the label ℓ, and a side condition that states that the operational specification only includes instances of rule (res) for those labels that do not have c as a bound atom.

The convenient use of freshness premises in the rules for π-calculus indicates that having freshness tests in rules is not only useful in modelling in a direct way meta-level operations such as substitutions and α-conversion, see Section 4, but it is also useful in the modelling of specific features of languages.

We denote by Π the set of π-calculus terms of [48], which is generated by the following grammar:

$$p, q ::= \mathbf{0} \mid \tau.p \mid \overline{a}b.p \mid a(b).p \mid \nu a.p \mid p + q \mid p \,||\, q \mid \ !p.$$

The encoding $[\![\cdot]\!]^\pi$ is a map from Π into terms of our nominal π-calculus and is defined as follows.

$$\begin{aligned}
[\![\mathbf{0}]\!]^\pi &= \mathbf{0} \\
[\![\tau.p]\!]^\pi &= \tau.[\![p]\!]^\pi \\
[\![\overline{a}b.p]\!]^\pi &= out(a, b, [\![p]\!]^\pi) \\
[\![a(b).p]\!]^\pi &= in(a, [b][\![p]\!]^\pi) \\
[\![\nu a.p]\!]^\pi &= \nu([a][\![p]\!]^\pi) \\
[\![p + q]\!]^\pi &= [\![p]\!]^\pi + [\![q]\!]^\pi \\
[\![p \,||\, q]\!]^\pi &= [\![p]\!]^\pi \,||\, [\![q]\!]^\pi \\
[\![!p]\!]^\pi &= ![\![p]\!]^\pi
\end{aligned}$$

Since we use a different notation for actions, the encoding is extended to labels as follows.

$$\begin{aligned}
[\![\tau]\!]^\pi &= \tau A \\
[\![ab]\!]^\pi &= inA(a, b) \\
[\![\overline{a}b]\!]^\pi &= outA(a, b) \\
[\![\overline{a}(b)]\!]^\pi &= boutA(a, [b])
\end{aligned}$$

For ease of reference, we repeat below the rules giving the semantics of the original early π-calculus from [48, Table 1.5, page 38]. We recall that, in those rules, $names(\ell)$ denotes the set of names that occur in the label ℓ, $bn(\ell)$ denotes

the set of bound names that occur in the label ℓ and $fn(p)$ denotes the set of names that have a free occurrence in the term p; see [48]. The formal definitions of these sets are repeated in Appendix C together with some other useful definitions from the standard theory of the π-calculus.

$$\frac{}{\tau.x \xrightarrow{\tau} x}\ (\tau\mathrm{O}) \qquad \frac{}{\overline{a}b.x \xrightarrow{\overline{a}b} x}\ (\mathrm{outO}) \qquad \frac{}{a(b).x \xrightarrow{ac} x[c/b]}\ (\mathrm{inO})$$

$$\frac{x_1 \xrightarrow{\ell} y_1}{x_1 + x_2 \xrightarrow{\ell} y_1}\ (\mathrm{sumO1}) \qquad bn(\ell) \cap fn(x_2) = \emptyset\ \frac{x_1 \xrightarrow{\ell} y_1}{x_1 \,||\, x_2 \xrightarrow{\ell} y_1 \,||\, x_2}\ (\mathrm{parO1})$$

$$\frac{x_2 \xrightarrow{\ell} y_2}{x_1 + x_2 \xrightarrow{\ell} y_2}\ (\mathrm{sumO2}) \qquad bn(\ell) \cap fn(x_1) = \emptyset\ \frac{x_2 \xrightarrow{\ell} y_2}{x_1 \,||\, x_2 \xrightarrow{\ell} x_1 \,||\, y_2}\ (\mathrm{parO2})$$

$$\frac{x_1 \xrightarrow{\overline{a}b} y_1 \quad x_2 \xrightarrow{ab} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau} y_1 \,||\, y_2}\ (\mathrm{comO1}) \qquad b \notin fn(x_2)\ \frac{x_1 \xrightarrow{\overline{a}(b)} y_1 \quad x_2 \xrightarrow{ab} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau} \nu b.(y_1 \,||\, y_2)}\ (\mathrm{closeO1})$$

$$\frac{x_1 \xrightarrow{ab} y_1 \quad x_2 \xrightarrow{\overline{a}b} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau} y_1 \,||\, y_2}\ (\mathrm{comO2}) \qquad b \notin fn(x_1)\ \frac{x_1 \xrightarrow{ab} y_1 \quad x_2 \xrightarrow{\overline{a}(b)} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau} \nu b.(y_1 \,||\, y_2)}\ (\mathrm{closeO2})$$

$$\frac{x \xrightarrow{\ell} y}{!x \xrightarrow{\ell} y \,||!x}\ (\mathrm{replO}) \qquad \frac{x \xrightarrow{\overline{a}b} y_1 \quad x \xrightarrow{ab} y_2}{!x \xrightarrow{\tau} (y_1 \,||\, y_2) \,||!x}\ (\mathrm{repl-comO})$$

$$b \notin fn(x)\ \frac{x \xrightarrow{\overline{a}(b)} y_1 \quad x \xrightarrow{ab} y_2}{!x \xrightarrow{\tau} (\nu b.(y_1 \,||\, y_2)) \,||!x}\ (\mathrm{repl-closeO})$$

$$a \neq z\ \frac{x \xrightarrow{\overline{z}a} y}{\nu a.x \xrightarrow{\overline{z}(a)} y}\ (\mathrm{openO}) \qquad c \notin names(\ell)\ \ \frac{x \xrightarrow{\ell} y}{\nu c.x \xrightarrow{\ell} \nu c.x}\ (\mathrm{resO})$$

Fig. 5. Rules for the early π-calculus

The reader should not confuse the operation of substitution over π-terms employed in the rule (inO) with the syntactic substitution over nominal terms defined in Section 4.1. Also, π-calculus terms are considered up to α-equivalence; so each term is 'equal' to all its α-equivalent ones. The definitions of substitution and α-equivalence of the π-calculus are standard and not provided in the main body of the paper. (They are, however, given in Appendix C for the sake of completeness.)

We now have all the ingredients to state a correspondence between the two calculi. The following theorem establishes that our formulation of the early π-calculus is operationally correct with respect to its original formulation.

Theorem 5 (Operational Correspondence: Early π-calculus). Luca: Check the actual formulation of this result. *For all $p, q \in \Pi$, $p \xrightarrow{\ell} q \Leftrightarrow [\![p]\!]^\pi \xrightarrow{[\![\ell]\!]^\pi} [\![q]\!]^\pi$, where ℓ ranges over the labels of the form τ, ab, $\overline{a}b$ and $\overline{a}(b)$ from the original early pi-calculus.*

The proof of Theorem 5 is contained in Appendix F.

5.3 A remark on the Barendregt Convention

Typically, in calculi with binders, terms are assumed to make use of some discipline on the choice of the names for the variables used in programs. This discipline is the so called *Barendregt Convention* and roughly states that, within a term, the names of bound variables must be chosen to be all different and also different from every free variable. For instance, in the context of the λ-calculus, the term $(\lambda x.(x\ y)\ x)$ is not considered part of the λ-calculus by the Barendregt Convention. This term is considered only in one of its α-equivalent forms that does respect the convention, for instance the term $(\lambda z.(z\ y)\ x)$.

This discipline on the names for variables is very useful in the development of the theory of calculi with binders. In particular it prevents the user from dealing with technicalities due to name clashes and allows one to focus directly on the computational aspects of a calculus. Of course, when implementing a calculus, the Barendregt Convention must, however, be addressed with a pre-processing step.

Our formulations of the lazy λ-calculus and of the early π-calculus do not make use of this convention; in fact, we do not make use of *any* convention regarding the name of atoms used in the terms.

Considering, for instance, our formulation of the lazy λ-calculus, the user can write the term $(\lambda([a].(\lambda([b].a\ b)))\ b)$. This term obviously does not adhere to the Barendregt Convention, as the atom b is used both as bound and free. The reader may notice however that this is not a problem in our formulation of the λ-calculus. Indeed, the term is automatically converted to an α-equivalent *good* one at the moment of performing a computational step. A transition $(\lambda([a].(\lambda([b].a\ b)))\ b) \to \lambda([c].b\ c)$ is indeed provable by rule (app), for all atoms c that are fresh in the subterm $(a\ b)$ of $\lambda([b].a\ b))$. In more detail, what happens in this situation is that the premise $y_0 \xrightarrow{a \overset{T}{\mapsto} x_1} y_1$ of the rule (app) cannot be satisfied in general when y_0 is instanciated for the term $(\lambda([b].a\ b)))\ b)$. Indeed, without α-conversion, the term $(\lambda([b].a\ b)))\ b)$ does not perform a transition $\xrightarrow{a \overset{T}{\mapsto} b}$ since the bound atom b is not fresh in the argument term b, and the premise $a\#z$ of $(abs1_{\mathrm{Ts}})$, instanciated in that context as $b\#b$, would not be satisfied. Fortunately, we set the term-for-atom substitution transitions to be up to α-equivalence, so one instance of the rule

$$\frac{x \approx_\alpha y \quad y \xrightarrow{a \overset{T}{\mapsto} b} z}{x \xrightarrow{a \overset{T}{\mapsto} b} z}\ (a \overset{T}{\mapsto} b \cdot \mathrm{upTo}\alpha).$$

is actually employed. The term $(\lambda([b].a\ b)))\ b)$ is thus α-converted to another term, which actually allows the premise $a\#z$ of $(abs1_{\mathrm{Ts}})$ to be satisfied.

The nominal machinery employed here plays a crucial role in this scenario, allowing for an implicit search for a suitable fresh atom, which is clearly guaranteed to exist.

In some sense, Nominal SOS naturally implements a sort of lazy Barendregt Convention, i.e. the change of atom names into suitable ones is performed during the computational steps, on demand, and only when facing the name clashes. All of this is possible thanks to the adoption of the nominal approach.

In implementations, these nominal calculi formalized using Nominal SOS do not require any pre-processing step in order to change the name of atoms in programs. This is particularly desirable in distributed contexts, where different pieces of code may come from different locations, written by different programmers who have no idea about the names for bound atoms used by others, and still their programs must be combined together.

For the sake of clarity, we point out that, by not assuming the Barengret Convention, the user can also write terms such as $(\lambda([a].(\lambda([a].a)\ a)\ b)$. In this case the computational step behaves as expected. In particular we can prove the transition $(\lambda([a].(\lambda([a].a)\ a)\ b) \rightarrow (\lambda([a].a)\ b)$, where the term $\lambda([a].a)$ remains unchanged because of the transition $\lambda([a].a) \xrightarrow{a \overset{T}{\mapsto} b} \lambda([a].a)$ proved using rule $(abs2_{\mathrm{Ts}})$.

6 Nominal bisimilarity

Very often, in the theory of calculi with binders, the ordinary bisimilarity is not a satisfactory equivalence. This is typical in calculi that allow terms to perform transitions whose labels mention fresh or bound variables. As a prominent example, we show the following example taken from [48] in the context of the π-calculus. We first recall the definition of bisimilarity over π-calculus processes, for which we overload the symbol $\leftrightarrow$.

Definition 15 (Bisimilarity) Bisimilarity $\leftrightarrow$ *is the largest symmetric binary relation* $\sim$ *on* Π *such that whenever* $P \sim Q$*, for all labels* l *it holds that if* $P \xrightarrow{l} P'$ *then there exists* Q'*, such that* $Q \xrightarrow{l} Q'$ *and* $P' \sim Q'$*.*

With respect to bisimilarity, the processes

$$P = \nu z.\overline{x}z.$$
$$Q = \nu z.(\ \overline{x}z.\ ||\ \nu w.\overline{w}y.\ )$$

are distinguished[5]. Indeed, since processes in the π-calculus are considered up to α-equivalence, we have that $P \xrightarrow{\overline{x}(y)}$. On the other hand Q can not turn

[5] The process $\nu w.\overline{w}y.$ performs an output on a channel which is not known by the external environment, therefore this process is bisimilar to $\mathbf{0}$.

its binder $x(z)$ into $x(y)$ by α-conversion, because y is one of its free variables. Therefore $Q \overset{\overline{x}(y)}{\not\to}$. Of course, P and Q should not be distinguished, and what actually happens in the theory of the π-calculus is that the transitions of the form $\overset{\overline{x}(z)}{\to}$ from P and Q are matched only for those variables z that are free in *both* P and Q. The bisimulation game is thus modified, and not all of the transitions from P and Q are considered in the matching process[6].

In what follows, we define this modified type of bisimilarity, here called *nominal bisimilarity*. In doing so, the reader should bear in mind that Nominal SOS models labels as open terms. In our framework, we therefore require that the bisimilarity would match labels containing abstractions only when the bound atoms are fresh in both of the considered terms. For instance, the bisimilarity in our formulation of the π-calculus will try to match bound output transitions of P and Q only for labels of the form $bout(a, [b]\mathbf{0})$ where the atom b is fresh in both P and Q.

Definition 16 (Nominal bisimilarity) *Let T be an NTSS.* Nominal bisimilarity $\leftrightarrows_T$ *is the largest symmetric binary relation $\sim$ over closed terms of T such that whenever $P \sim Q$, for all labels l such that for all $a \in ba(l)$, $a\#P$, $a\#Q$, it holds that if $P \xrightarrow{l} P'$ then there exists Q', such that $Q \xrightarrow{l} Q'$ and $P' \sim Q'$.*

In what follows we will always omit the subscript in $\leftrightarrows_T$, and write $\leftrightarrows$ for nominal bisimilarity, as T will be always clear from the context.

We prove that what the nominal bisimilarity does in the ordinary π-calculus is exactly what bisimilarity does in our formulation of the π-calculus when ignoring the matching of substitution transitions in the bisimulation game. We denote this equivalence with $\underline{\leftrightarrow}^{-}$. In what follows the encoding $[\![\cdot]\!]^{\pi}$ is the mapping defined in Section 5.2.

Theorem 6 (Nominal bisimilarity is bisimilarity, when ignoring substitutions). *For all $P, Q \in \Pi$, $P \leftrightarrows Q$ if, and only if, $[\![P]\!]^{\pi} \underline{\leftrightarrow}^{-} [\![Q]\!]^{\pi}$.*

The proof of Theorem 6 can be found in Section I.

The reader may wonder what is the equivalence over π-calculus terms that corresponds to nominal bisimilarity, including substitution transitions. The next theorem provides us with an answer: nominal bisimilarity in our formulation of the early π-calculus coincides with Sangiorgi's open bisimilarity, see [48, Section 4.2] and [47]. The open bisimilarity involves substitutions, which are defined as follows, as recalled from [48] (Definition 1.1.3 on page 14).

Definition 17 (Substitutions involved in the open bisimilarity) *A* substitution σ *is a function on names that is the identity except on a finite set.*

In this section we use the symbol σ for substitutions, as it is standard in the literature on the π-calculus. The reader should not however confuse substitutions with the sorts of Definition 1.

We now recall the definition of open bisimilarity from [48, 47].

[6] This point is explained carefully on pages 64-65 of [48].

Definition 18 (Open Bisimilarity) Open bisimilarity $\underline{\leftrightarrow}^o$ *is the largest symmetric relation* $\sim$ *on* Π *such that whenever* $P \sim Q$*, and* σ *is a substitution (of Definition 17), if* $P\sigma \xrightarrow{\alpha} P'$*, then there exists* Q'*, such that* $Q\sigma \xrightarrow{\alpha} Q'$ *and* $P' \sim Q'$*.*

The reader should notice that this definition is the very basic formulation of open bisimilarity, which does not involve distinctions, see [48] and [47]. In Definition 18, it is important to note that the ranging over all the substitutions is performed at each step of the bisimulation game.

Note furthermore that the substitutions involved in the open bisimilarity are capture avoiding. This is a necessary constraint, for open bisimilarity would not be a satisfactory equivalence otherwise. The reader can indeed consider the two terms

$$P = \nu a.\overline{b}b.$$
$$Q = \nu c.\overline{b}b.$$

and the substitution σ that maps the name b to a and is the identity over all the other names. If we allowed the application of substitutions to capture free names, by applying σ to the two terms above we have that $P\sigma = \nu a.\overline{a}a.$, which is nominal bisimilar to $\mathbf{0}$, and $Q\sigma = \nu c.\overline{a}a.$, which can perform an output on the channel a. In this scenario, the terms P and Q would be thus distinguished, even though they are α-equivalent.

Theorem 7 (Open bisimilarity and Bisimilarity coincide). *For all* $P, Q \in \Pi$*,* $P \underline{\leftrightarrow}^o Q$ *if, and only if,* $[\![P]\!]^\pi \leftrightarrow [\![Q]\!]^\pi$*.*

The proof of Theorem 7 can be found in Section J.

In passing, we briefly discuss the version of open bisimilarity that involves *distinctions*. Indeed, the open bisimilarity as in Definition 18 is not a satisfactory equivalence for the π-calculus, as it does not take into account the fact that two atoms that are bound by a restriction ν can never be identified. This point is made clear on page 167 of [48] by means of an example which we shall now show. Let us consider the two processes P and Q defined as follows[7]:

$$P = \nu z.\nu w.\overline{x}z.\overline{x}w. \quad (\overline{z} \,||\, w)$$
$$Q = \nu z.\nu w.\overline{x}z.\overline{x}w. \quad (\overline{z}.w + w.\overline{z}).$$

We have that P and Q are not open bisimilar. To see this, it suffices to note that the subterm $(\overline{z} \,||\, w)$ of P and the subterm $(\overline{z}.w + w.\overline{z})$ of Q are not open bisimilar. Indeed, if we fix a substitution σ that maps w to z and is the identity over all the other channel names, we have that $(\overline{z} \,||\, w)\sigma \xrightarrow{\tau}$ and $(\overline{z}.w + w.\overline{z})\sigma \not\xrightarrow{\tau}$.

However, both the channel names z and w occur underneath a ν restriction and they are thus forced to be distinct. Therefore, no substitution can make the two channels suddenly able to communicate with each other.

[7] In order to present our point more clearly, the two processes $(\overline{z} \,||\, w)$ and $(\overline{z}.w + w.\overline{z})$ employ a CCS-style syntax.

The processes P and Q should indeed be considered equivalent. Sangiorgi equips the open bisimilarity with distinctions in [47], see also [48], that keep track of the names that cannot be identified during the bisimulation game. The open bisimilarity with distinctions is a satisfactory equivalence for the π-calculus, as argued in [48].

However, the scenario discussed above is too dependent on the intrinsic meaning of binders to be captured by a general theory like the one we put forward in this paper. For instance, in the π-calculus there are two binders $\nu a.P$ and $a(b).P$, but only the former populates distinctions. The latter binder has a different meaning. Since this work addresses only a basic and uniform account of binders, providing an adaptation of the nominal bisimilarity of Definition 16 in order to tackle distinctions, although possible, is not considered in this first development and is part of our future work.

7 Applicative Bisimilarity

In the context of the lazy λ-calculus, one of the most interesting notions of bisimilarity is the *applicative bisimilarity* due to Samson Abramsky, [2]. Below we recall the definition of this equivalence. We also recall that a λ-term is closed if it contains no free variables, and that we denote the set of closed λ-terms by Λ^0.

Definition 19 (Applicative Bisimilarity in the λ-calculus) *The* applicative bisimilarity *is the largest symmetric relation $\simeq$ on Λ^0 such that whenever $M \simeq N$, if $M \to \lambda a.M'$ for some variable a and $M' \in \Lambda$, then there exist some variable b and $N' \in \Lambda$ such that*

- *$N \to \lambda b.N'$, and*
- *$M'[P/a] \simeq N'[P/b]$, for all $P \in \Lambda^0$.*

It is important to remark that the applicative bisimilarity of the λ-calculus is defined over closed λ-terms. Indeed, this equivalence is strongly unsatisfactory over terms that contain free variables. For instance, for any variables a, b and c, it holds that $a \simeq b$ and $\lambda a.b \simeq \lambda a.c$.

The notion of applicative bisimilarity can be accommodated within the context of Nominal SOS in the obvious way.

Definition 20 (Applicative Bisimilarity over $\mathbb{C}(\Sigma^\lambda)$) *The* applicative bisimilarity *is the largest symmetric relation $\simeq$ on $\mathbb{C}(\Sigma^\lambda)^0$ such that whenever $M \simeq N$, if $M \to \lambda([a].M')$ for some atom a and $M' \in \mathbb{C}(\Sigma^\lambda)$, then there exist some atom b and $N' \in \mathbb{C}(\Sigma^\lambda)$ such that*

- *$N \to \lambda([b].N')$, and*
- *$M'[P/a] \simeq N'[P/b]$, for all $P \in \mathbb{C}(\Sigma^\lambda)^0$.*

The following theorem states that the applicative bisimilarity of our formulation of the lazy λ-calculus and that of its original formulation coincide.

Theorem 8 (Applicative Bisimilarity in Λ and in Σ^λ coincide). *For all $M, N \in \Lambda^0$, $M \simeq N$ if, and only if, $[\![M]\!]^\lambda \simeq [\![N]\!]^\lambda$.*

The proof of Theorem 8 follows easily from Theorem 4, which states the operational correctness of our lazy λ-calculus with respect to the original calculus.

It is now natural to wonder what are the relationships between the applicative and the nominal bisimilarity in our formulation of the lazy λ-calculus. Unfortunately, nominal bisimilarity is a very unsatisfactory equivalence over $\mathbb{C}(\Sigma^\lambda)^0$. In particular, all the terms in $\mathbb{C}(\Sigma^\lambda)^0$ that have a normal form are equated.

Theorem 9 (Nominal Bisimilarity equates all 'terminating' terms). *For all $M, N \in \mathbb{C}(\Sigma)^0$, if M and N have a normal form then $M \leftrightarrow N$.*

The proof of Theorem 9 can be found in Section L.

Notice moreover, that both nominal and applicative bisimilarity equate all the terms that do not have a normal form. The implications of Definitions 16 and 20 are indeed vacuously true. From this fact and from Theorem 9, the following theorem easily follows.

Theorem 10 (Applicative is included in the nominal bisimilarity). *It holds that $\simeq \subset \leftrightarrow$.*

However, the applicative bisimilarity is not included in the nominal bisimilarity. This is stated in the following theorem.

Theorem 11 (Nominal is *not* included in the applicative bisimilarity). *It holds that $\leftrightarrow \not\subset \simeq$.*

Theorem 11 follows easily from the fact that applicative bisimilarity distinguishes some binding-closed terms that have a normal form. Any two such terms are counter-example witnesses for Theorem 11. By way of example, let us consider the two terms $M = \lambda([a].a)$ and $N = \lambda([a].(a\ a))$. Notice first that M and N are binding-closed and they have a normal form. By Theorem 9 we have therefore that $M \leftrightarrow N$. However, it holds that $M \not\simeq N$. To see this, the reader should notice that after the 'parameter passing' of the term N, the two terms are distinguished. Namely, at the second step of the applicative bisimilarity we are required to check, among other checks, whether the two terms $a[N/a] = \lambda([a].(a\ a))$ and $(a\ a)[N/a] = (\lambda([a].(a\ a))\ \ \lambda([a].(a\ a)))$ are applicative bisimilar. This is not the case, as the former has a normal form while the latter does not have a normal form. We have that $\lambda([a].(a\ a)) \not\simeq (\lambda([a].(a\ a))\ \ \lambda([a].(a\ a)))$ and consequently $M \not\simeq N$.

From the fact that the applicative bisimilarity of our lazy λ-calculus coincides with that of its original formulation (Theorem 8), the following theorem follows as a straightforward consequence of Theorems 10 and 11.

Theorem 12 (Nominal and Applicative Bisimilarity relation through the encoding). *For all $M, N \in \Lambda^0$,*

- *$M \simeq N$ implies $[\![M]\!]^\lambda \leftrightarrow [\![N]\!]^\lambda$, and*
- *$[\![M]\!]^\lambda \leftrightarrow [\![N]\!]^\lambda$ does* not *imply $M \simeq N$.*

7.1 Characterizing applicative bisimilarity in terms of nominal bisimilarity

In this section we present a variation on the semantics of the nominal lazy λ-calculus from Section 5.1 that ensures that the nominal bisimilarity and the applicative bisimilarity coincide.

The semantics employs the signature Σ^{λ} as given in Section 3 and also repeated in Section 5.1. The semantics of this formulation of the lazy λ-calculus contains the transition $\rightarrow$ with the same meaning as in Section 5.1. It also contains transitions $\xrightarrow{P}$, where P is a binding-closed term. For terms M, N and a binding-closed λ-term P, a transition $M \xrightarrow{P} N$ can be read as *the term M progresses to N when it is applied to the argument P*. The rules for term-for-atom substitution transitions, as described in Section 4.1, are also part of the semantics. The following rules define our variant of the formulation of the lazy λ-calculus.

$$\frac{}{\lambda([a]x) \rightarrow \lambda([a]x)}\ (\text{abs1AP}) \qquad \frac{x_0 \xrightarrow{a \overset{T}{\mapsto} y} x_1 \qquad \forall b.(b\#y)}{\lambda([a]x_0) \xrightarrow{y} x_1}\ (\text{abs2AP})$$

$$\frac{x_0 \rightarrow y_0 \qquad y_0 \xrightarrow{x_1} y_1 \qquad y_1 \rightarrow y_2}{(x_0\ x_1) \rightarrow y_2}\ (\text{app1AP})$$

$$\frac{(x_0\ x_1) \rightarrow y_1 \qquad y_1 \xrightarrow{x_2} y_2}{(x_0\ x_1) \xrightarrow{x_2} y_2}\ (\text{app2AP})$$

Moreover, the transition relations $\rightarrow$, $\xrightarrow{P}$ for any binding-closed term P, and the term-for-atom substitution transitions are up to α-equivalence. Recall that, by Definition 14, this means that the set of rules for the language contains also the rules for α-conversion transitions and the rules for atom-for-atom substitution transitions. The rules for atom-for-atom substitution transitions are also set to be up to α-equivalence.

It is worth noting that the formulation above of the lazy λ-calculus performs the parameter passing only of binding-closed terms. To see this, the reader can notice that the premises $\forall b.(b\#y)$ of rule (abs2AP), which stand for the infinite conjunction of premises $\{a\#y \mid a \in C\}$[8], are simultaneously satisfied only when the term instanciated for y does not contain free atoms, i.e. it is binding-closed. If a term contains free atoms then one of the premises would not be satisfied and the rule would not be applicable. So to be clear, in a transition $M \xrightarrow{P} N$, the term P must be binding-closed.

Restricting the parameter passing only to binding-closed terms makes things simpler when proving that applicative and nominal bisimilarity coincide and is in line with the definition of applicative bisimilarity. Consider for instance two

[8] We recall that, in Definition 7, the indexing sets I, J and K are possibly infinite.

λ-terms M and N and let us have that $M \simeq N$. If we were to remove the premises $\forall b.(b\#y)$ from rule (abs2AP), then we would admit also transitions of the form $M \xrightarrow{P} N$, with P a term that is possibly not binding-closed. To prove $[\![M]\!]^\lambda \leftrightarrows [\![N]\!]^\lambda$, we must thus have that the behaviours of M and N match also over those transitions. However, from $M \simeq N$ we can only conclude that M and N behave the same when the argument P is in Λ^0, see Definition 19, and therefore that $[\![M]\!]^\lambda$ and $[\![N]\!]^\lambda$ behave the same only for those transitions $\xrightarrow{P}$ where P is in $\mathbb{C}(\Sigma)^0$.

The reader is also invited to notice that, in this formulation of the lazy λ-calculus, the treatment of the λ-abstraction is very similar to the way the early π-calculus treats the input prefix operator. The purpose of both these two binders is indeed very similar: both accept an argument from the external environment. However, their treatment in the classic formulations of the λ- and π-calculus is rather different. In the λ-calculus, the semantic rules detect the need for an argument passing by detecting syntactically the λ-abstraction involved, as in the premise $x_0 \to \lambda a.y_0$ of the rule (appO) of Section 5.1. Other formulations of λ-calculi typically act in the same way. For instance, they are usually based on the classic β-rule

$$(\lambda a.M\ N) \to M[N/a],$$

which syntactically detects the presence of an abstraction as left operand of an application.

In the early π-calculus, the input prefix operator has instead the capability to progress spontaneously. With the rule (inO) of Section 5.2, namely,

$$a(b).x \xrightarrow{ac} x[c/b],$$

the process progresses as if the channel name c has been actually passed.

The reader can see that in the formulation above, by means of rule (abs2AP), we formulated the λ-calculus following the approach of the π-calculus. The same spontaneous progress is possible from abstractions. Rule (app2AP) allows applications to be fed with arguments, too. After the normal form of an application is calculated, the argument passing and the consequent progress take place. Rule (app2AP) is necessary for the nominal bisimilarity to equate, for instance, abstractions with applications. Consider indeed the identity function $\lambda([a]a)$ and the application $\lambda([b]\lambda([a]a))\ c$ which reduces to the identity function. A reasonable equivalence relation might want to consider these two terms equivalent. However, if we omitted (app2AP) the nominal bisimilarity would distinguish the two terms, as the term $\lambda([a]a)$ can perform transitions of the form $\xrightarrow{P}$ that the term $\lambda([b]\lambda([a]a))\ c$ cannot match.

The following theorem states that in sharp contrast to Theorem 12, applicative bisimilarity in the lazy λ-calculus coincides with nominal bisimilarity in the nominal formulation of the lazy λ-calculus given above.

Theorem 13 (Applicative Bisimilarity in Λ and Nominal Bisimilarity in the new formulation of λ coincide). *For all $M, N \in \Lambda^0$, $M \simeq N$ if, and only if, $[\![M]\!]^\lambda \leftrightarrows [\![N]\!]^\lambda$.*

The proof of Theorem 13 can be found in Section M.

8 Related Work

Nominal SOS is not the only approach studied so far in the literature that aims at a uniform treatment of binders and names in the operational semantics of programming and specification languages. We are aware of a number of existing approaches that accommodate variables and binders inside the SOS framework, namely those by Fokkink and Verhoef in [15], by Middelburg in [30, 31], by Miller and Tiu in [33] and by Fiore and Staton in [13] (originally, by Fiore and Turi [14]). Moreover, Gacek, Miller and Nadathur, in [20], Lakin and Pitts in [27] (MLSOS) and also in [28, 29, 26] (αML), and Sewell at al. in [49] have also proposed formalization of binders within SOS-like frameworks. In the following sections we discuss in some detail the approaches that are more related to our work.

8.1 MLSOS and αML

In [27], Lakin and Pitts provide the meta-language MLSOS in which the various transition relations of SOS can be specified in a functional style language. In this section we do not repeat the definitions of the system from [27]. However, a reader who is familiar with functional programming will have no difficulty in understanding the code fragments we present.

In [27], the authors provide a formulation of the λ-calculus that employs the parallel reduction strategy, i.e. the reduction is performed in any context. Below we show a formulation of the simpler call-by-name λ-calculus, see [6]. Intuitively, in the call-by-name λ-calculus the reduction is inductively shared only to the first argument of an application. Namely, if $M \to M'$, then $(M\ N) \to (M'\ N)$. The π-calculus is not formalized in [27] and therefore we do not provide code for it. The example for the λ-calculus suffices to show to the reader how MLSOS provides a language for the specification of the SOS semantics of calculi with binders.

```
1: nametype var;;
2: datatype lam =
3:       Var of var
4:       Lam of <<var>> lam
5:       App of lam * lam;;
6:
7: let rec sub x t t' = narrow t' as
8:       Var y:  ((x=:=y);t or ((x=/=y : var);t')
9:       Lam <<a>>t'':   Lam <<a>>sub x t t'':
10:      App (t_1 t_2):  App ((sub x t t_1) (sub x t t_2));;
11:
12: let rec beta (t_1 t_2) = narrow (t_1 t_2) as
13:      (t t):          yes
```

```
14:      (App (t_1 t_2), t_3):    beta(t_1 t_2)
15:      (App ((Lam <<a>>t_1), t_2), t'):
16:              some t_1', t_2': lam in
17:              sub(a, t_2', t_1') =:= t';;
```

Lines 1-5 define the grammar for λ-terms, lines 7-10 define the substitution procedure and lines 12-17 defines the reduction relation. The reader can see how the way of defining the terms of a calculus recalls the way of defining inductive types in a functional language. Also, the definition of the various transition relations are encoded as some sort of function definitions of functional languages.

It is important however to notice that, in MLSOS, the only operations allowed on atoms are equality tests and the generation of fresh atoms. In MLSOS, in a λ-term

$$\lambda\langle\langle a\rangle\rangle(Var\ a)$$

the variable a actually ranges over the infinite set of atoms, and will be instanciated to a fresh atom on demand. The user cannot name atoms directly.

This approach stems directly from Nominal Logic [41]. In particular, the nominal И-quantifier is employed in this case, see also [16] where it was introduced. This solution to the modeling of languages was adopted already in FreshML, [51, 50]. FreshML adds to ML the capability to generate fresh new names with the keyword **fresh**. For instance, in the term $let\ x\ =$ **fresh** $in\ Lam\langle x\rangle(Var\ x)$, the occurrences of the keywork **fresh** are evaluated by generating a name never seen before and placed in the body of the abstraction. The management of this, and the actual atom names employed in the execution of a program, are hidden to the user.

A different choice is instead made in the meta-language αML, [28, 29], which, in a sense, is what MLSOS evolved into. The syntax of αML shares many similarities with the one of MLSOS, in particular, it stems from its functional language style syntax. However, some peculiarities of αML also come from the world of the constraint logic programming, [25]. The management of variables of αML follows a different philosophy. As argued in Lakin's Thesis [26], where an excellent and complete overview of the language αML can be found, in the term

$$Lam\langle\langle x\rangle\rangle(Lam\langle\langle y\rangle\rangle(Var\ x))$$

the variables x and y are not assumed to be distinct just for having a distinct name. This term can denote both the λ-terms $\lambda a.\lambda b.a$ and $\lambda a.\lambda a.a$, as y is not required to denote a name that is different to the one denoted by x.

One can however state explicitly some conditions on name. In particular, the user can disseminate the program with constraints of name equality and freshness. If we wish the variables x and y to denote different names in the term above, we can rewrite it in αML as

$$\exists x : var.\exists y : var.(x\#y\ \&\ Lam\langle\langle x\rangle\rangle(Lam\langle\langle y\rangle\rangle(Var\ x))).$$

The management of constraints is delegated to constraint solving facilities of constraint logic programming. The interested reader is invited to refer to Lakin's Thesis [26].

The reader must notice that the handling of variables in all of the mentioned approaches (MLSOS, FreshML and αML) is different from how Nominal SOS handles name. In particular, in those approaches the user does not have access to concrete atoms/variables directly. This goes along with what in the nominal context is called *the equivariance property*, see [16] and especially [41]. This property roughly states that the meaning of programs should not depend upon the concrete atom name which is chosen in the implementation of abstractions. On the other hand, within Nominal SOS one can write programs whose behaviour is strongly associated to a particular bound name. For instance, let us consider augmenting the rules of our formulation of the lazy λ-calculus with the following rule, where N is a given closed term and a is a fixed atom. (The following rule is therefore not replicated for all atoms and closed terms.)

$$\frac{x \xrightarrow{a \mapsto N} x'}{\lambda([a]x) \to x'}$$

In this case, the calculus acts non-deterministically for λ-abstraction terms that use the atom a as bound. Namely, given a closed term M, the term $\lambda([a]M)$ can perform two transitions. We can apply the rule (abs) of Section 5.1 in order to prove the standard transition $\lambda([a]M) \to \lambda([a]M)$. On the other hand, we can apply the rule above. Applying this rule, the abstraction $\lambda([a]M)$ pretends that the parameter passing of a term N took place. Since the rule above is defined only for the atom a, the rule above cannot be applied for a term $\lambda([b]M)$, with a and b distinct atoms.

Defining the behaviour of programs in a way that it depends on specific bound names should be considered bad language design. In Nominal SOS, however, a language specifier is free to explore also this possibility.

Another difference between Nominal SOS and the approaches of MLSOS and αML is that the latter ones provide for a primitive built-in support for managing the abstract syntax trees of terms which are identified up to α-equivalence. The motivation for such a choice is that this support for α-equivalence is required in any calculus that contains binders. If α-equivalence is not built-in, this support needs to be coded up by hand. In our context, the code for this management does not represent much of a problem; indeed it can be automatically generated, as shown in Section 4. Secondly, as our design choice, we prefer α-equivalence to be a transition in the semantics just like any other, so that it can be the subject of meta-theorems based on the shape of rules that may be developed in the future.

8.2 $FO\lambda^{\Delta\nabla}$

In [33], Miller and Tiu make use of a logic that is already equipped with λ-terms; this approach is called *the λ-tree approach* to encoding syntax. The particular logic they employ is called $FO\lambda^{\Delta\nabla}$ (fold-nabla). The main peculiarity of this logic is that, besides containing the connectives and quantifiers of ordinary logic,

terms can be of the form $\lambda y.P$, meaning that the variable y is bound in the body P. Also the notion of application as higher-order term passing is already embedded in the logic. The logic is equipped with a sequent calculus that takes care of this λ-calculus style features and also deals with the technicalities that arise with dealing with binders. Another distinctive feature of the logic $FO\lambda^{\Delta\nabla}$ is the use of the *new quantifier* ∇. Basically the meaning of a formula $\nabla x.\Phi$ is that x is a fresh new name within the scope of the quantifier ∇, i.e. within the formula Φ.

A user can use the logic in order to specify the semantics of calculi with binders. In [33], the authors formulate the late and early π-calculus. The semantics of these calculi are presented conveniently by a set of rules. For instance, the rule

$$\frac{P \xrightarrow{\alpha} R}{P + Q \xrightarrow{\alpha} R}$$

is expressed by the $FO\lambda^{\Delta\nabla}$ formula $(P \xrightarrow{\alpha} R) \supset P + Q \xrightarrow{\alpha} R$. This formula formalizes a part of the behaviour of the choice operator in much the same way as the rule (sum1) does in our formulation of Section 5.2. The symbol $\supset$ stands for the standard implication, P,Q and R are variables ranging over terms, $\xrightarrow{\alpha}$ is a binary relation symbol[9] in the signature of the logic and $+$ is a binary function symbol in the signature of the logic. The whole set of rules that define the late and the early π-calculus can be found in [33]. Here we discuss only a few rules that highlight the characteristic features of the approach. Consider for instance the following rule

$$\frac{\nabla x.(Px \xrightarrow{A} Qx)}{\nu x.(Px) \xrightarrow{A} \nu x.(Qx)}\ (\text{parRes1})$$

Thanks to the quantifier ∇, the premise of the rule is satisfied when the process that instanciates the variable P performs a transition when the name x in its body is chosen fresh. If this happens then the term $\nu x.P$ can progress accordingly. The quantifier ∇ is thus very expressive, and offers the possibility to treat names as fresh new ones when needed. This feature turns out to be frequently useful, especially in the context of the π-calculus and similar calculi.

The reader may want to consider also the rule for the input prefix

$$\frac{}{in(\lambda y.M) \xrightarrow{\downarrow XU} (MU)}\ (\text{in})$$

In this rule X and U represent variables for channel names. The computational step for input prefix processes is formalized by means of the λ-calculus style parameter passing embedded in the logic.

The sequent calculus for $FO\lambda^{\Delta\nabla}$ takes care of deriving the expected formulae, dealing with the binders with its λ-calculus style features and with special

[9] In [33], relations also are labeled with a type. In order to ease the presentation we here omit type information.

management for the quantifier ∇. The sequent calculus is not showed here, the interested reader can consult [33] and [32] for it.

The main difference between Nominal SOS and the approach in [33] is that in Nominal SOS no technicality concerning the use of binders is hidden and given as built-in. Moreover, Nominal SOS does not rely on existing notions for parameter passing. The language specifier needs to define nearly everything. This may give the user also the possibility to explore, for instance, other types of substitution or equivalence of terms.

8.3 SOS in Abella

In [20], Gacek, Miller and Nadathur describe a method to specify calculi with binders. They make use of the λ-tree syntax approach, discussed in the previous section about the approach with $FO\lambda^{\Delta\nabla}$. The aim of the authors is to use the proof assistant Abella, [18], in order to reason about calculi with binders. Abella is a proof assistant for the specification logic G, [19], which we do not discuss here. In the approach of [20], the authors do not make use of the logic G directly; they instead provide a second specification logic that is more suited for their purposes. This latter logic is the theory of the intuitionistic second-order hereditary Harrop formulae, there called hH^2. The logic hH^2 turns out to be a convenient vehicle in order to formulate rule-based definitions such as the ones encountered in SOS. It also turns out that, since hH^2 is a subset of $\lambda Prolog$, [39], specifications in hH^2 can be computed and executed effectively.

The authors provide an encoding from formulae of hH^2 into the logic G, making the use of the Abella proof assistant possible.

In [20], the authors formalize the simply typed call-by-value λ-calculus, see [7]. Roughly speaking, in the call-by-value strategy, the λ-calculus parameter passing involved in the application $(\lambda x.M)\ N$ is performed only when the term N is a value, i.e. when N is an abstraction. The term N is thus first evaluated until it becomes a value. For the sake of the presentation, we show the code for the call-by-value λ-calculus in its untyped version.

∀ a,r [value(abs a r)]

∀ m,n,m' [step m m' ⊃ step(app m n) step(app m' n)]

∀ m,n,n' [value m ∧ step n n' ⊃ step(app m n) step(app m n')]

∀ a, r, m [value m ⊃ step(app(abs a r) m) (r m)]

The predicate 'value' recognizes values, i.e. abstractions. The term $(app\ m\ n)$ corresponds to the application $(M\ N)$. In the formulation of [20], the term $(abs\ a\ (\lambda x.r\ x))$ represents an abstraction in the programming language in which the bound variable x has type a. The predicate 'step' is defined by the last three implications in the expected way.

Since the theory proposed in [20] follows the λ-tree approach, its differences with Nominal SOS are by and large the same stated in the previous section, concerning the approach with $FO\lambda^{\Delta\nabla}$. As another difference, we can see that the logic hH^2 lacks an explicit and clear mechanism for stating freshness conditions. Note, however, that the specification logic G of Abella is in general more powerful and, in particular, it is capable of expressing freshness conditions. When

necessary, the language specifier can modify the code produced by the encoding from hH^2 into the logic G in order to state freshness conditions.

8.4 SOS with Ott

In [49], Sewell et al. describe Ott, a tool support for the specification of calculi with binders in SOS. The general idea behind Ott is similar to the one for the approach described in the previous section for SOS specifications in Abella. The authors provide a meta language for formulating the semantics of calculi with binders and the purpose of the Ott tool support is to generate from the specification the code for some popular proof assistants. In this approach, the proposed meta-language is not a logic as in the case of the previous section; it is instead a simple and intuitive language for which we shall give an example below.

Again, we do not describe the entire system, but the reader can obtain an idea of how this system works by means of an example. In [49], the authors formalize the call-by-value λ-calculus. As the language tends to be particularly verbose, below we present our formulation of the simpler call-by-name λ-calculus. The code below has been also stripped of some decorating syntax and those parts that are not relevant to understand Ott at this stage. The interested reader is invited to read [49] for a full description of the system and the syntax employed.

```
grammar
term t :: 't_' ::=
        x                 ::      var
        \x . t            ::      lam (+ bind x in t +)
        t t'              ::      app

terminals :: 'terminals_' ::=
        \         ::      lambda {{tex \lambda}}
(1)     -->       ::      red {{tex \longrightarrow}}

defn
t_1 --> t_2 ::reduce:: {{com [[t_1]] reduces to [[t_2]] } }


------------------------------ :: ax
\x . t_1 t_2 --> {t_2/x}t_1

t_1 --> t_1'
------------------------------ :: rule1
t_1 t --> t_1' t
```

The first part of the code is devoted to the formalization of the terms of the calculus. It is important to note that the language provided by Ott requires the user to use the special decoration *(+ bind x in t +)* in order to specify binding information. The rest is quite simple to understand. In particular, the language

provided by Ott allows the user to specify the SOS semantic rules for the calculus at hand as she/he would *draw* them from the paper to the text editor.

The tool support Ott takes the specification and returns code for the most popular interactive theorem provers, such as Coq [4], HOL [43] and Isabelle/HOL [40]. Nicely, Ott also returns code for a LaTeX presentation of the calculus. The reader can see that the listing above is indeed disseminated every now and then with presentational information. For instance, with the line that we labelled (1), the latex code generated will be such that the transition relation denoted $--{>}$ in the text editor will be represent by the symbol $\longrightarrow$.

Nominal SOS and the approaches discussed in the previous sections employ a single binding. Roughly speaking, they are abstractions in the style of λ-calculus where in the term $\lambda x.M$ only one variable can be bound. Ott allows for more complex binding structures, for instance structured patterns, multiple mutually recursive let definitions, and dependent record patterns, see [49] and [23].

However, the single binding employed by Nominal SOS and by the other frameworks has proved to be expressive enough for most of the cases.

8.5 General Remarks

As a general remark on the related work, the mentioned systems attack the problem of defining the operational semantics of nominal calculi using an approach that is different from the one proposed in this paper. Our long-term goal is to develop systematically a meta-theory of SOS for calculi with binders and, following the lead of the meta-theory of ordinary SOS, to investigate at a syntactic level those semantic phenomena that are specifically concerned with binding.

From this point of view, the previous approaches do not seem close enough to the standard framework of ordinary SOS, while Nominal SOS, being a slight variation of it with specific primitive notions for dealing with binders, is fairly close. We believe that the close relationship between Nominal SOS and ordinary SOS will have the following benefits.

1. We will be able to lift/adapt already existing meta-theory to the context of binders with relative little effort. Transporting the body of meta-theoretic results to the contexts of the other approaches seems to be a more difficult task.
2. The content of the meta-theorems and the line of investigation will resemble closely those achieved so far for ordinary SOS, and with which SOS users are familiar.
3. Moreover, thanks to the nominal approach, we hopefully have an intuitive and familiar language with which we could explain why certain calculi afford a property while others do not, for instance by means of presence/absence of freshness premises, or of a syntactic discipline in the use of them or of other related nominal concepts.

Carrying out a study of the meta-theory of languages with binders based on the other approaches is still possible, but it is not part of our immediate future research goals.

9 Conclusions and Future works

In this paper, we have introduced a framework, called Nominal SOS, for modelling the operational semantics of nominal calculi. The framework comes equipped with the basic features used in defining such calculi, namely, substitution and α-conversion. Since these notions are generated from the nominal signature they can be replaced by, and the user can experiment with, for instance, other notions of substitution. Our syntax has two levels of variables like the one in [53], but we do not take permutations as primitive—because there is no need to do so; we obtain α-conversion 'for free' based on atom-for-atom substitutions as a nominal SOS theory. This suffices for our examples of interest, e.g. the λ- and π-calculi.

We used the framework to specify the semantics of the lazy λ-calculus and early π-calculus and showed that our formulations of the semantics coincide with the original ones. A notion of nominal bisimilarity arises naturally from our framework. Moreover, we showed that the notion of nominal bisimilarity in our semantics of the early π-calculus coincides with open bisimilarity in the original semantics.

Nominal SOS provides a framework to extend the meta-theory of SOS to the nominal setting. This paper contains only the basic developments of the framework, accompanied by some examples. The framework seems close enough to the formalization of the standard theory of SOS in such a way that the mode of operation for carrying out meta-theory over Nominal SOS and the content of the corresponding meta-theorems would resemble very closely those presented for instance in [3, 38]. Hence, lifting previous results from the meta-theory of standard SOS to the new context seems also feasible. Our main aim is to develop the theory and applications of Nominal SOS so that it reaches a level of maturity that is comparable to that of the theory of classic SOS, as surveyed in, e.g., [3, 38]. More specifically, the main goals of our future work are as follows.

- We intend to provide further evidence that Nominal SOS is expressive enough to capture the original semantics of nominal calculi, such as variants of the π-calculus and its higher-order version [46], the psi-calculi [8] and the object calculi [1], and to prove formally the correspondence between the presentation in terms of Nominal SOS and the original ones. Also, we plan to address different notions of equivalence betweens terms. Just to name a few examples, it would be worth defining within Nominal SOS a notion that is the analogous of the applicative bisimilarity in the context of the λ-calculus, [2], and investigating the relation between the notion of nominal bisimilarity and the applicative bisimilarity. Also, an adaptation of the nominal bisimilarity that coincides with the open bisimilarity with distinctions, [48, 47], when applied to π-calculus terms.
- We plan to develop the meta-theory of Nominal SOS, for example by providing congruence formats ifor behavioural semantics n the context of calculi with binders, possibly generalizing those proposed in [55] and [13], for instance. Also, *confluence* is an important property that has not been tackled yet by the theory of SOS in the context of rule formats. Many important

confluence results stem from the realm of calculi equipped with binders; the reader may indeed think of the λ-calculus and its variants. It would be thus desirable to provide rule formats guaranteeing the confluence property within the framework of Nominal SOS. Meta-theory over Nominal SOS can be carried out also for all those phenomena that are specifically related to binders. For instance it would be worth providing rule formats guaranteeing that the late and early bisimilarity, see [48], coincide.
- We expect to extend a wealth of classic SOS meta-results and techniques to the framework of Nominal SOS.
- We plan on providing tool support for Nominal SOS.

9.1 Extensions of the framework

In a sense, in this paper we have formulated Nominal SOS in its most basic form. We are aware of a few possible extensions of the framework, which would be worth adding.

Negative freshness premises. A possible extension of the framework would be to add the possibility to have premises of rules of the form $\neg(a\#t)$, with t a term. Intuitively, this premise is satisfied when the atom a is *not* fresh in the term t. Negative freshness tests as premises do not seem to be employed in general in the definition of nominal calculi, and thus they are left out in our formulation of Section 3.

Variables in bound terms. It would be worth extending the syntax of nominal terms of Section 2 by augmenting the grammar with the following form of terms

$$t ::= \ldots \mid ([x_A]t_\sigma)_{[A]\sigma}$$

Intuitively, x is a variable of atom sort. The reader may recall that the terms defined in Section 2 are such that abstractions are only of form $[a].t$ with a concrete atom a. It is important to notice that, given a term t, the term $[x].t$ is an open term while the term $[a].t$ is closed. This addition should be accompanied by an extension of the set of possible premises in rules so that also premises of the form $x\#t$, with x a variable of atom sort are allowed. By way of example, these two extensions would allow Nominal SOS to formulate the semantics of the lazy λ-calculus with the following two rules.

$$\frac{}{\lambda([z]x) \to \lambda([z]x)} \qquad \frac{x_0 \to \lambda([z]y_0) \quad y_0 \overset{z \overset{T}{\mapsto} x_1}{\longrightarrow} y_1 \quad y_1 \to y_2}{x_0\ x_1 \to y_2}$$

The rules above use the variable z in order to range over the set of atoms. These two rules alone can replace the rules (abs) and (app) of Section 5.1, which use concrete atoms and they are thus replicated for every atom. The semantics of the λ-calculus would be thus finitely formalized.

Specific syntax to state the meaning of binders. Another possible extension of the Nominal SOS framework is to augment the signature of an NTSS with information that concerns the meaning of the binders in the language. The meaning of binders used in nominal calculi can indeed be of various nature. For instance, the binder λ in the formulation of the λ-calculus binds its argument atom in a way that supports a substitution for it in a parameter passing fashion. On the other hand, the meaning of the binder ν in the formulation of the π-calculus binds its argument atom to indicate that its name must be considered private.

It would be worth investigating a suitable language that allows the user to express how binders are intended to be used in the calculus. Insofar this subject is concerned, we have no preliminary developments so far.

References

1. Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer-Verlag, New York, 1996.
2. Samson Abramsky. The lazy lambda calculus. In D.A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison Wesley, Reading, Mass., 1990.
3. Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. Elsevier, 1999.
4. The Coq Proof Assistant. http://coq.inria.fr/.
5. Jos C. M. Baeten and Cornelis A. Middelburg. *Process Algebra with Timing*. Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, Berlin, 2002.
6. Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics, Revised edition*. North Holland, 1984.
7. Henk. P. Barendregt. *Lambda calculi with types*, pages 117–309. Oxford University Press, Inc., New York, NY, USA, 1992.
8. Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proceedings of the 24th Annual IEEE Symposium on Logic In Computer Science*, pages 39–48, Washington, DC, USA, 2009. IEEE Computer Society.
9. Karen L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In *LICS*, pages 153–164, 1998.
10. Christiano de O. Braga, Edward H. Haeusler, José Meseguer, and Peter D. Mosses. Mapping modular sos to rewriting logic. In *Proceedings of the 12th international conference on Logic based program synthesis and transformation*, LOPSTR'02, pages 262–277, Berlin, Heidelberg, 2003. Springer-Verlag.
11. Matteo Cimini, Mohammad Reza Mousavi, Michel A. Reniers, and Murdoch James Gabbay. Nominal SOS. *Electronic Notes in Theoretical Computer Science*, 286:103–116, 2012.
12. Maribel Fernandez and Murdoch J. Gabbay. Nominal rewriting. *Information and Computation*, 205(6):917–965, 2007.
13. Marcelo Fiore and Sam Staton. A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 49–58, Washington, DC, USA, 2006. IEEE Computer Society.

14. Marcelo P. Fiore and Daniele Turi. Semantics of name and value passing. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 93–104. IEEE Computer Society, Los Alamiots, CA, USA, 2001, 2001.
15. Wan Fokkink and Chris Verhoef. A conservative look at operational semantics with variable binding. *Information and Computation*, 146:24–54, October 1998.
16. Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, Washington, 1999.
17. Murdoch James Gabbay and Aad Mathijssen. A nominal axiomatization of the lambda calculus. *Journal of Logic and Computation*, 20(2):501–531, 2010.
18. Andrew Gacek. The abella interactive theorem prover (system description). In *Proceedings of the 4th international joint conference on Automated Reasoning*, IJCAR '08, pages 154–161, Berlin, Heidelberg, 2008. Springer-Verlag.
19. Andrew Gacek, Dale Miller, and Gopalan Nadathur. Combining generic judgments with recursive definitions. In *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 33–44, Washington, DC, USA, 2008. IEEE Computer Society.
20. Andrew Gacek, Dale Miller, and Gopalan Nadathur. Reasoning in Abella about structural operational semantics specifications. In A. Abel and C. Urban, editors, *International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2008)*, number 228 in Electronic Notes in Theoretical Computer Science, pages 85–100, 2008.
21. D. Plotkin Gordon. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.
22. Jan F. Groote, Michel A. Reniers, Jos J. Van Wamel, and Mark B. Van Der Zwaag. Completeness of timed mcrl. *Fundamenta Informaticae*, 361:2002, 2002.
23. Ott User Guide. http://www.cl.cam.ac.uk/ pes20/ott/ott_manual_0.20.3.html.
24. Pieter H. Hartel. Letos - a lightweight execution tool for operational semantics. *Software - Practice and Experience*, 29:1379–1416, December 1999.
25. Joxan Jaffar, Michael Maher, Kim Marriott, and Peter Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1):1–46, 1996.
26. Matthew R. Lakin. *An executable meta-language for inductive definitions with binders*. PhD thesis, University of Cambridge, 2010.
27. Matthew R. Lakin and Andrew M. Pitts. A metalanguage for structural operational semantics. In *Symposium on Trends in Functional Programming*, pages 1–16, 2007.
28. Matthew R. Lakin and Andrew M. Pitts. Resolving inductive definitions with binders in higher-order typed functional programming. In G. Castagna, editor, *18th European Symposium on Programming (ESOP '09)*, volume 5502 of *Lecture Notes in Computer Science*, pages 47–61. Springer, Mar 2009.
29. Matthew R. Lakin and Andrew M. Pitts. Encoding abstract syntax without fresh names. *Journal of Automated Reasoning*, 2011. To appear.
30. Cornelis A. Middelburg. Variable binding operators in transition system specifications. *Journal of Logic and Algebraic Programming*, 47(1):15–45, 2001.
31. Cornelis A. Middelburg. An alternative formulation of operational conservativity with binding terms. *Journal of Logic and Algebraic Programming (JLAP)*, 55(1-2):1–19, 2003.
32. Dale Miller and Alwen Tiu. A proof theory for generic judgments: An extended abstract. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 118–127, Washington, DC, USA, 2003. IEEE Computer Society.

33. Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic*, 6:749–783, October 2005.
34. Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
35. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part i. *Information and Computation*, 100, 1989.
36. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, ii. *Information and Computation*, 100:41–77, September 1992.
37. Mohammad Reza Mousavi and Michel A. Reniers. Prototyping sos meta-theory in maude. *Electronic Notes in Theoretical Computer Science*, 156:135–150, May 2006.
38. Mohammad Reza Mousavi, Michel A. Reniers, and Jan F. Groote. Sos formats and meta-theory: 20 years after. *Theoretical Computer Science*, 373(3):238–272, 2007.
39. Gopalan Nadathur and Dale Miller. An overview of lambda prolog. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming (ICLP-SLP 1988)*, pages 810–827. MIT Press, 1988.
40. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.
41. Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186(2):165–193, 2003.
42. Gordon D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
43. The HOL project. http://hol.sourceforge.net/.
44. Teodor C. Przymusinski. Well-founded semantics coincides with three-valued stable semantics. *Fundamenta Informaticae*, XIII:445–463, 1990.
45. Davide Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996.
46. Davide Sangiorgi. pi-calculus, internal mobility, and agent-passing calculi. In *Selected papers from the 6th international joint conference on Theory and practice of software development*, TAPSOFT '95, pages 235–274, Amsterdam, The Netherlands, The Netherlands, 1996. Elsevier Science Publishers B. V.
47. Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33(1):69–97, 1996.
48. Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge Univ. Press, 2001.
49. Peter Sewell, Francesco Z. Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar, and Rok Strniša. Ott: effective tool support for the working semanticist. *ACM SIGPLAN Notices*, 42:1–12, October 2007.
50. Mark R. Shinwell. *The Fresh Approach: functional programming with names and binders*. PhD thesis, University of Cambridge, 2005.
51. Mark R Shinwell, Andrew M. Pitts, and Murdoch J. Gabbay. FreshML: Programming with binders made simple. In *Eighth ACM SIGPLAN International Conference on Functional Programming (ICFP 2003), Uppsala, Sweden*, pages 263–274. ACM Press, 2003.
52. Bent Thomsen. A theory of higher order communicating systems. *Information and Computation*, 116(1):38–57, 1995.
53. Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323:473–497, 2004.

54. Rob J. van Glabbeek. The meaning of negative premises in transition system specifications II. *Journal of Logic and Algebraic Programming (JLAP)*, 60-61:229–258, 2004.
55. Axelle Ziegler, Dale Miller, and Dale Palamidessi. A congruence format for name-passing calculi. In *Proceedings of the 2nd Workshop on Structural Operational Semantics (SOS'05)*, volume 156 of *Electronic Notes in Theoretical Computer Science*, pages 169–189, Lisbon, Portugal, 2005. Elsevier Science B.V.

A Useful definitions for nominal terms

We recall the syntax for nominal terms given in Definition 2.

$$t, u ::= \langle\rangle \mid x \mid a \mid ([a]t) \mid f(t, \ldots, t_n) \mid \langle t, u\rangle.$$

We define $\mathcal{V}(t)$, the set of variables that occur in the term t, by induction on the structure of t.

$$\begin{aligned}
\mathcal{V}(\langle\rangle) &= \emptyset \\
\mathcal{V}(x) &= \{x\} \\
\mathcal{V}(a) &= \emptyset \\
\mathcal{V}([a]t) &= \mathcal{V}(t) \\
\mathcal{V}(f(t_1, \ldots, t_n)) &= \mathcal{V}(t_1) \cup \cdots \cup \mathcal{V}(t_n) \\
\mathcal{V}(\langle t, u\rangle) &= \mathcal{V}(t) \cup \mathcal{V}(u)
\end{aligned}$$

We define $\mathcal{A}(t)$, the set of atoms that occur in the term t, by induction on the structure of t.

$$\begin{aligned}
\mathcal{A}(\langle\rangle) &= \emptyset \\
\mathcal{A}(x) &= \emptyset \\
\mathcal{A}(a) &= \{a\} \\
\mathcal{A}([a]t) &= \{a\} \cup \mathcal{A}(t) \\
\mathcal{A}(f(t_1, \ldots, t_n)) &= \mathcal{A}(t_1) \cup \cdots \cup \mathcal{A}(t_n) \\
\mathcal{A}(\langle t, u\rangle) &= \mathcal{A}(t) \cup \mathcal{A}(u)
\end{aligned}$$

We define $ba(t)$, the set of bound atoms that occur in the term t, by induction on the structure of t.

$$\begin{aligned}
ba(\langle\rangle) &= \emptyset \\
ba(x) &= \emptyset \\
ba(a) &= \emptyset \\
ba([a]t) &= \{a\} \cup ba(t) \\
ba(f(t_1, \ldots, t_n)) &= ba(t_1) \cup \cdots \cup ba(t_n) \\
ba(\langle t, u\rangle) &= ba(t) \cup ba(u)
\end{aligned}$$

We define $fa(t)$, the set of atoms a in $\mathcal{A}(t)$ that have an occurrence in t that is not within the scope of an abstraction $[a]\_$, by induction on the structure of t.

$$
\begin{aligned}
fa(\langle\rangle) &= \emptyset \\
fa(x) &= \emptyset \\
fa(a) &= \{a\} \\
fa([a]t) &= fa(t) \setminus \{a\} \\
fa(f(t_1, \ldots, t_n)) &= fa(t_1) \cup \cdots \cup fa(t_n) \\
fa(\langle t, u\rangle) &= fa(t) \cup fa(u)
\end{aligned}
$$

B Useful definitions for λ-terms

We define $FV(M)$, the set of free variables that occur in a λ-term M, by induction on the structure of M.

$$
\begin{aligned}
FV(a) &= \{a\} \\
FV(\lambda a.M) &= FV(M) \setminus \{a\} \\
FV(M\ N) &= FV(M) \cup FV(N)
\end{aligned}
$$

We say that a λ-term M is *closed* if $FV(M) = \emptyset$, i.e., if M does not contain free variables.

Given a variable a and two λ-terms M and N, we define the *substitution* operation $M[N/a]$ defined inductively on the structure of M. In what follows a and b are different variables.

$$
\begin{aligned}
a[N/a] &= N \\
b[N/a] &= b \\
(\lambda a.M)[N/a] &= \lambda a.M \\
(\lambda b.M)[N/a] &= \lambda b.(M[N/a]), \quad \text{if } b \notin FV(N) \\
(M_1\ M_2)[N/a] &= (M_1[N/a]\ M_2[N/a])
\end{aligned}
$$

We also recall that α-*equivalence* in the λ-calculus is defined as the least congruence $\equiv$ over λ-terms such that if $b \notin FV(M)$ then $\lambda a.M \equiv \lambda b.(M[b/a])$. In the λ-calculus, α-equivalent terms are considered syntactically equal and the substitution operation defined above is total. For example, if $c \neq b$ and $c \neq a$,

$$(\lambda b.a)[b/a] = \lambda c.b,$$

because $\lambda b.a \equiv \lambda c.a$.

Note that $M[b/a]$ is defined (without any need for α-conversion) if, and only if,

- either $a = b$
- or $a \neq b$ and every subterm of M of the form $\lambda b.M'$ is within the scope of a $\lambda a._-$.

More generally, $M[N/a]$ is defined (without any need for α-conversion) if, and only if, every subterm of M of the form $\lambda b.M'$ for some $b \in FV(N)$, is either within the scope of a $\lambda a._-$ or is such that $b = a.$.

C Useful definitions for π-calculus terms

We recall that a π-calculus label α can have one of the following forms:

$$\alpha ::= \tau \mid \overline{a}b \mid ab \mid \overline{a}(b)$$

where a and b are two channel names.

We define $names(\alpha)$, the set of names that occur in the label α, as follows.

$$\begin{aligned} names(\tau) &= \emptyset \\ names(\overline{a}b) &= \{a, b\} \\ names(ab) &= \{a, b\} \\ names(\overline{a}(b)) &= \{a, b\} \end{aligned}$$

The set $bn(\alpha)$ of names that are bound in the label α is defined as follows.

$$\begin{aligned} bn(\tau) &= \emptyset \\ bn(\overline{a}b) &= \emptyset \\ bn(ab) &= \emptyset \\ bn(\overline{a}(b)) &= \{b\} \end{aligned}$$

We define $fn(p)$, the set of names that occur free in the process p, as follows.

$$\begin{aligned} fn(\mathbf{0}) &= \emptyset \\ fn(\tau.p) &= fn(p) \\ fn(\overline{a}b.p) &= \{a, b\} \cup fn(p) \\ fn(a(b).p) &= \{a\} \cup (fn(p) \setminus \{b\}) \\ fn(\nu a.p) &= fn(p) \setminus \{a\} \\ fn(p + q) &= fn(p) \cup fn(q) \\ fn(p \,||\, q) &= fn(p) \cup fn(q) \\ fn(!p) &= fn(p) \end{aligned}$$

Given a π-calculus term p and names a and b, we define the *substitution* operation $p[b/a]$ inductively on the structure of p. In what follows a, b, c and d range over names. Substitution acts in the expected way over names; in particular $a[b/a] = b$ and $c[b/a] = c$ when $c \neq a$.

$$\begin{aligned} \mathbf{0}[b/a] &= \mathbf{0} \\ (\tau.p)[b/a] &= \tau.(p[b/a]) \\ (\overline{c}d.p)[b/a] &= \overline{c[b/a]}d[b/a].p[b/a] \\ (c(d).p)[b/a] &= c[b/a](d).(p[b/a])) \quad \text{with } d \neq a,\ d \neq b \\ (c(a).p)[b/a] &= c[b/a](a).p \\ (\nu c.p)[b/a] &= \nu c.(p[b/a]) \quad \text{with } \ c \neq a, c \neq b \\ (\nu a.p)[b/a] &= \nu a.p \\ (p + q)[b/a] &= p[b/a] + q[b/a] \\ (p \,||\, q)[b/a] &= p[b/a] \,||\, q[b/a] \\ (!p)[b/a] &= !(p[b/a]) \end{aligned}$$

We also recall that α-*equivalence* is defined in the π-calculus as the least congruence $\equiv$ over π-calculus terms such that

- if $c \notin fn(p)$ then $a(b).p \equiv a(c).(p[c/b])$ and
- if $b \notin fn(p)$ then $\nu a.p \equiv \nu b.(p[b/a])$.

As in the λ-calculus, α-equivalent terms are considered syntactically equal in the π-calculus and the substitution operation defined above is total up to $\equiv$.

D Proof of Theorem 2

Let T be a $NTSS$. Consider a closed term t over the signature of T and an atoms a, b. We prove the two implications in the statement of Theorem 2 separately.

If $t \xrightarrow{a \overset{A}{\mapsto} b} t'$ *then* $t' = t[b/a]$. Suppose that $t \xrightarrow{a \overset{A}{\mapsto} b} t'$. We prove that $t' = t[b/a]$ by induction on the proof of the transition $t \xrightarrow{a \overset{A}{\mapsto} b} t'$. We proceed by a case analysis on the last rule used in the proof of that transition and consider explicitly only three selected cases.

- Case $t = a \xrightarrow{a \overset{A}{\mapsto} b} b = t'$ by rule ($\mathrm{a1_{As}}$) in Figure 1 on page 11. In this case, $t' = a[b/a]$ and we are done.
- Case $t = [c]u \xrightarrow{a \overset{A}{\mapsto} b} [c]u' = t'$ by rule ($\mathrm{abs1_{As}}$) because, by a shorter inference, $u \xrightarrow{a \overset{A}{\mapsto} b} u'$, $c \neq a$ and $c \neq b$. In this case, the induction hypothesis yields $u' = u[b/a]$. Since $c \neq a$ and $c \neq b$, we have that

  $$t[b/a] = ([c]u)[b/a] = [c](u[b/a]) = [c]u' = t',$$

  and we are done.
- Case $t = f(t_1, \ldots, t_n) \xrightarrow{a \overset{A}{\mapsto} b} f(t'_1, \ldots, t'_n) = t'$ by rule ($\mathrm{f_{As}}$) because, by shorter inferences, $t_i \xrightarrow{a \overset{A}{\mapsto} b} t'_i$ for each $1 \leq i \leq n$. By the inductive hypothesis, we have that $t'_i = t_i[b/a]$ for each $1 \leq i \leq n$. So, $t' = f(t_1[b/a], \ldots, t_n[b/a]) = t[b/a]$, which was to be shown.

If $t' = t[b/a]$ *then* $t \xrightarrow{a \overset{A}{\mapsto} b} t'$. This implication can be shown by induction on the structure of t. The details are similar to those of the above-given proof and therefore we omit them.

E Proof of Theorem 4: Operational correspondence for the lazy λ-calculus

The encoding $[\![\cdot]\!]^{\lambda}$ is the map from Λ into terms of the nominal λ-calculus defined in Section 5.1, which we repeat here for ease of reference.

$$\begin{aligned}[\![a]\!]^\lambda &= a\\ [\![\lambda a.M]\!]^\lambda &= \lambda([a][\![M]\!]^\lambda)\\ [\![MN]\!]^\lambda &= [\![M]\!]^\lambda [\![N]\!]^\lambda\end{aligned}$$

In the remainder of this section, we use $\equiv$ to denote syntactic equality up to α-equivalence of the lazy λ-calculus.

Our proof of Theorem 4 relies on the following lemmas, stating the correctness of α-conversion and substitution transitions for the λ-calculus.

Lemma 4 (Correctness of α-conversion transitions).

- *For all $M, N \in \Lambda$, if $M \equiv N$ then $[\![M]\!]^\lambda \approx_\alpha [\![N]\!]^\lambda$.*
- *For all $M \in \Lambda$ and $N' \in \mathbb{C}(\Sigma^\lambda)$, if $[\![M]\!]^\lambda \approx_\alpha N'$ then there exists some $N \in \Lambda$ such that $M \equiv N$ and $[\![N]\!]^\lambda = N'$.*

Lemma 5 (Correctness of substitution transitions). *For all $M, N \in \Lambda$, atom a and $M' \in \mathbb{C}(\Sigma^\lambda)$, the following statements hold.*

- *If $[\![M]\!]^\lambda \xrightarrow{a \overset{T}{\mapsto} [\![N]\!]^\lambda} M'$ then $M' = [\![M'']\!]^\lambda$ for some $M'' \equiv M[N/a]$.*
- *If $M[N/a] \equiv M'$ then $[\![M]\!]^\lambda \xrightarrow{a \overset{T}{\mapsto} [\![N]\!]^\lambda} [\![M']\!]^\lambda$.*

The proofs of Lemmas 4 and 5 are lengthy and can be found in Appendix E.1.

We show the two implications in the statement of Theorem 4 separately.

Proof of the left-to-right implication: if $M \to N$ then $[\![M]\!]^\lambda \to [\![N]\!]^\lambda$. We show the above statement by induction on the length of the proofs of provable transitions in the original λ-calculus. We proceed by a case analysis on the last rule used in the proof of the transition $M \to N$. Recall that λ-calculus terms are considered up to α-equivalence.

- Suppose that $M \equiv \lambda a.M' \to \lambda a.M' \equiv N$ has been shown with rule (absO), for some M'. By the definition of $[\![\cdot]\!]^\lambda$, we have that $[\![\lambda a.M']\!]^\lambda = \lambda([a][\![M']\!]^\lambda)$. By means of rule (abs) we can prove the transition

  $$\lambda([a][\![M']\!]^\lambda) \to \lambda([a][\![M']\!]^\lambda).$$

  Observe now that $[\![M]\!]^\lambda \approx_\alpha \lambda([a][\![M']\!]^\lambda)$ and $\lambda([a][\![M']\!]^\lambda) \approx_\alpha [\![N]\!]^\lambda$ both hold, by Lemma 4 and the definition of $[\![\cdot]\!]^\lambda$. Since the transition relation $\to$ over $\mathbb{C}(\Sigma^\lambda)$ is up to α-equivalence, we have that $[\![M]\!]^\lambda \to [\![N]\!]^\lambda$ and we are done.
- Suppose that $M \to N$ has been shown with rule (appO) because, for some closed terms M_1, M_2 and M_3,
  - $M \equiv M_1\ M_2$,
  - $M_1 \to \lambda a.M_3$ and
  - $M_3[M_2/a] \to N$.

By Lemma 5, we have that

$$[\![M_3]\!]^\lambda \xrightarrow{a \overset{T}{\mapsto} [\![M_2]\!]^\lambda} [\![M_3[M_2/a]]\!]^\lambda.$$

The inductive hypothesis and the definition of $[\![\cdot]\!]^\lambda$ yield that

$$[\![M_1]\!]^\lambda \to [\![\lambda a.M_3]\!]^\lambda = \lambda([a][\![M_3]\!]^\lambda)$$

and

$$[\![M_3[M_2/a]]\!]^\lambda \to [\![N]\!]^\lambda.$$

We can therefore use rule (app) in order to prove the transition

$$[\![M_1\ M_2]\!]^\lambda \to [\![N]\!]^\lambda.$$

By Lemma 4, $[\![M]\!]^\lambda \approx_\alpha [\![M_1\ M_2]\!]^\lambda$. Since the transition relation $\to$ over $\mathbb{C}(\Sigma^\lambda)$ is up to α-equivalence, we have that $[\![M]\!]^\lambda \to [\![N]\!]^\lambda$ and we are done.

Proof of the right-to-left implication: if $[\![M]\!]^\lambda \to N$ *then* $M \to N'$ *and* $[\![N']\!]^\lambda = N$*, for some term* N'. We show the above statement by induction on the length of the proof of the transition $[\![M]\!]^\lambda \to N$. We proceed by a case analysis on the last rule used in the proof.

- Suppose that $[\![M]\!]^\lambda \to N$ has been shown using rule (abs). By the definition of $[\![\cdot]\!]^\lambda$, this means that M is $\lambda a.M'$ for some M' and that $N = [\![M]\!]^\lambda = \lambda([a][\![M']\!]^\lambda)$. By rule (absO), we have that $M \to M$ and we are done.
- Suppose that $[\![M]\!]^\lambda \to N$ has been shown using rule (app). By the definition of $[\![\cdot]\!]^\lambda$, this means that M is M_1M_2 for some $M_1, M_2 \in \Lambda$ and that the following transitions are provable by shorter inferences:
  1. $[\![M_1]\!]^\lambda \to \lambda([a]M_1')$, for some M_1',
  2. $M_1' \xrightarrow{a \mapsto [\![M_2]\!]^\lambda} M_3$ and
  3. $M_3 \to N$.

  Our aim is to make use of the rule (appO) in order to prove a transition $M_1M_2 \to N'$ for some N' such that $[\![N']\!]^\lambda = N$. To this end, observe, first of all, that the inductive hypothesis applied to item 1 above yields that $M_1 \to N_1$ for some N_1 such that $[\![N_1]\!]^\lambda = \lambda([a]M_1')$. By the definition of $[\![\cdot]\!]^\lambda$, N_1 is $\lambda a.N_1'$ for some N_1' such that $[\![N_1']\!]^\lambda = M_1'$. So, we can rephrase the second item above thus:

  $$[\![N_1']\!]^\lambda \xrightarrow{a \mapsto [\![M_2]\!]^\lambda} M_3.$$

  By Lemma 5, we have that there is some M_3' such that $M_3' \equiv N_1'[M_2/a]$ and $M_3 = [\![M_3']\!]^\lambda$. Therefore we may apply the inductive hypothesis to item 3 above and the fact that λ-terms are considered equal up to $\equiv$ to obtain that $N_1'[M_2/a] \to N'$ for some N' such that $[\![N']\!]^\lambda = N$.

  In summary, we have that
  - M is M_1M_2 for some $M_1, M_2 \in \Lambda$,

- $M_1 \to \lambda a.N_1'$ for some N_1' and
- $N_1'[M_2/a] \to N'$ for some N' such that $[\![N']\!]^\lambda = N$.

We can now use rule (appO) to infer that $M_1 M_2 \to N'$ and we are done.

– Suppose that $[\![M]\!]^\lambda \to N$ has been shown using rule $(\langle\rangle \cdot \mathrm{upTo}\alpha)$. This means that, for some terms M_1 and M_2,
  - $[\![M]\!]^\lambda \approx_\alpha M_1$,
  - $M_1 \to M_2$ and
  - $M_2 \approx_\alpha N$.

  By Lemma 4, the first item above yields that there exists some $M_1' \in \Lambda$ such that $M \equiv M_1'$ and $[\![M_1']\!]^\lambda = M_1$. We may therefore use the inductive hypothesis to infer that $M_1' \to M_2'$ for some M_2' such that $[\![M_2']\!]^\lambda = M_2$. Using Lemma 4 again and the third item above, we have that there exists some $N' \in \Lambda$ such that $M_2' \equiv N'$ and $[\![N']\!]^\lambda = N$. In summary,

$$M \equiv M_1' \to M_2' \equiv N' \text{ and } [\![N']\!]^\lambda = N.$$

  Since terms in the λ-calculus are considered up to $\equiv$, it follows that $M \to N'$ and we are done.

E.1 Proof of Lemmas 4 and 5

The following lemma will be useful in the remainder of this section.

Lemma 6. *Given an atom a and a λ-term M, $a\#[\![M]\!]^\lambda$ if, and only if, a is fresh in M (that is, $a \notin FV(M)$).*

The size of a term $M \in \Lambda$, denoted by $|M|$, is its length in symbols. In some of the inductive proofs to follow, we will use the two simple, but important, properties of the size of terms mentioned below.

Lemma 7.

1. *For all $M, N \in \Lambda$, if $M \equiv N$ then $|M| = |N|$.*
2. *For all $M \in \Lambda$ and atoms a, b, the terms M and $M[b/a]$ have the same size.*

Since the rules defining $\approx_\alpha$ rely on the transition relation axiomatizing atom-for-atom substitutions, we prove the statements in Lemmas 4 simultaneously with two auxiliary results stating the correctness of atom-for-atom substitution transitions. For the sake of clarity, we summarize the four statements we prove simultaneously below.

Lemma 8.

1. *For all $M, N \in \Lambda$, if $M \equiv N$ then $[\![M]\!]^\lambda \approx_\alpha [\![N]\!]^\lambda$.*
2. *For all $M \in \Lambda$ and $N' \in \mathbb{C}(\Sigma^\lambda)$, if $[\![M]\!]^\lambda \approx_\alpha N'$ then there exists some $N \in \Lambda$ such that $M \equiv N$ and $[\![N]\!]^\lambda = N'$.*
3. *For all $M \in \Lambda$, atoms a, b and $M' \in \mathbb{C}(\Sigma^\lambda)$, if $[\![M]\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} M'$ then there exists some $N \in \Lambda$ such that $M[b/a] \equiv N$ and $M' = [\![N]\!]^\lambda$.*

4. For all $M, N \in \Lambda$ and atoms a, b, if $M[b/a] \equiv N$ then $[\![M]\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} [\![N]\!]^\lambda$.

Proof. We prove all the statements in the lemma simultaneously by induction on the size of M. We assume, as our induction hypothesis, that the four statements above hold for all λ-terms of size strictly smaller than n and we prove them for terms of size n. In the remainder of the proof, we write i_n, $i \in \{1, 2, 3, 4\}$, for statement i for terms of size n and establish each of these claims in turn.

Proof of statement 1_n. We proceed by an inner induction on the proof of $M \equiv N$, where M is a λ-term of size n, and a case analysis on the last rule used in such a proof. We limit ourselves to presenting the details of the proof for three selected cases. The others can be shown following similar lines.

- Assume that $M \equiv N$ has been shown using transitivity because $M \equiv M'$ and $M' \equiv N$, for some M', by shorter inferences. Since $|M| = |M'|$, we may use the inner inductive hypothesis twice to obtain that $[\![M]\!]^\lambda \approx_\alpha [\![M']\!]^\lambda$ and $[\![M']\!]^\lambda \approx_\alpha [\![N]\!]^\lambda$. Since $\approx_\alpha$ is transitive (Lemma 3), we may conclude that $[\![M]\!]^\lambda \approx_\alpha [\![N]\!]^\lambda$, and we are done.
- Assume that $M = \lambda a.M'$, $N = \lambda a.N'$ and $M' \equiv N'$ by a shorter inference. By the inductive hypothesis, $[\![M']\!]^\lambda \approx_\alpha [\![N']\!]^\lambda$. Since $\approx_\alpha$ is a congruence (Lemma 3), we have that

  $$[\![M]\!]^\lambda = \lambda([a][\![M']\!]^\lambda) \approx_\alpha \lambda([a][\![N']\!]^\lambda) = [\![N]\!]^\lambda,$$

  and we are done.
- Assume that $M = \lambda a.M'$, $N = \lambda b.(M'[b/a])$ and $b \notin FV(M')$. Since $|M'| < |M| = n$, we may use the inductive hypothesis for statement 4 to infer that

  $$[\![M']\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} [\![M'[b/a]]\!]^\lambda.$$

  As $b \notin FV(M')$, we also have that $b \# [\![M']\!]^\lambda$ by Lemma 6. Thus, using rule $(\mathrm{abs1}_\alpha)$ in Figure 2 on page 14,

  $$[a][\![M']\!]^\lambda \approx_\alpha [b][\![M'[b/a]]\!]^\lambda.$$

  Using the definition of $[\![\cdot]\!]^\lambda$ and the substitutivity of $\approx_\alpha$, it follows that $[\![M]\!]^\lambda \approx_\alpha [\![N]\!]^\lambda$.

Proof of statement 2_n. We proceed by an inner induction on the proof of $[\![M]\!]^\lambda \approx_\alpha N'$, where M is a λ-term of size n, and a case analysis on the last rule used in such a proof. We limit ourselves to presenting the details of the proof for some selected cases. The others can be shown following similar lines.

- Assume that $[\![M]\!]^\lambda \approx_\alpha N'$ because
  - $M = \lambda a.M_1$, and thus $[\![M]\!]^\lambda = \lambda([a][\![M_1]\!]^\lambda)$,
  - $N' = \lambda([a]N_1')$, for some N_1', and
  - $[\![M_1]\!]^\lambda \approx_\alpha N_1'$ by shorter inference.

Since $|M_1| < |M| = n$, we may use the inductive hypothesis to infer that there is some $N_1 \in \Lambda$ such that $[\![N_1]\!]^\lambda = N_1'$ and $M_1 \equiv N_1$. Let $N = \lambda a.N_1$. Then $M \equiv N$ and $[\![N]\!]^\lambda = N'$ both hold, and we are done.

– Assume that $[\![M]\!]^\lambda \approx_\alpha N'$ because
  - $M = \lambda a.M_1$, and thus $[\![M]\!]^\lambda = \lambda([a][\![M_1]\!]^\lambda)$,
  - $N' = \lambda([b]N_1')$, for some N_1' and b such that $b\#[\![M_1]\!]^\lambda$ and $[\![M_1]\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} N_1'$.

  Since $|M_1| < |M| = n$, we may apply the inductive hypothesis for statement 3 to infer that there is some N_1 such that $N_1 \equiv M_1[b/a]$ and $[\![N_1]\!]^\lambda = N_1'$. Let $N = \lambda b.N_1$. As $b\#[\![M_1]\!]^\lambda$, we also have that $b \notin FV(M_1)$ by Lemma 6. Therefore,

$$M = \lambda a.M_1 \equiv \lambda b.(M_1[b/a]) \equiv \lambda b.N_1 = N.$$

  Moreover, $[\![N]\!]^\lambda = N'$ and we are done.

– Assume that $[\![M]\!]^\lambda \approx_\alpha N'$ has been derived using transitivity because, by shorter inferences, $[\![M]\!]^\lambda \approx_\alpha N_1'$ and $N_1' \approx_\alpha N'$ for some N_1'. By the inner inductive hypothesis, we have that there is some $N_1 \in \Lambda$ such that $[\![N_1]\!]^\lambda = N_1'$ and $M \equiv N_1$. Since $|N_1| = |M| = n$, we again may apply the inner inductive hypothesis to obtain that there is some $N \in \Lambda$ such that $[\![N]\!]^\lambda = N'$ and $N_1 \equiv N$. By the transitivity of $\equiv$, we may now infer that $M \equiv N$ and we are done.

Proof of statement 3$_n$. We proceed by an inner induction on the proof of $[\![M]\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} M'$, where M is a λ-term of size n, and a case analysis on the last rule used in the proof. We limit ourselves to presenting the details of the proof for some selected cases. The others can be shown following similar lines.

– Assume that $M = \lambda c.M_1$ for some M_1 and some atom c such that $c \neq a$ and $c \neq b$. Then $[\![M]\!]^\lambda = \lambda([c][\![M_1]\!]^\lambda) \xrightarrow{a \overset{A}{\mapsto} b} M'$ has been derived using the instance of rule ($\mathrm{f_{As}}$) for the function symbol λ in Figure 1 on page 11. This means that, for some M_1',
  - $M' = \lambda([c]M_1')$ and
  - $[c][\![M_1]\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} [c]M_1'$.

  Since $c \neq a$, the transition $[c][\![M_1]\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} [c]M_1'$ has been derived using rule ($\mathrm{abs1_{As}}$) in Figure 1. This means that, by a shorter inference, $[\![M_1]\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} M_1'$. By induction, there is some N_1' such that $N_1' \equiv M_1[b/a]$ and $[\![N_1']\!]^\lambda = M_1'$. Let $N = \lambda c.N_1'$. Then, as $c \neq a$ and $c \neq b$,

$$M[b/a] = \lambda c.(M_1[b/a]) \equiv \lambda c.N_1' = N.$$

  Moreover, $[\![N]\!]^\lambda = M'$ and we are done.

– Assume that $[\![M]\!]^\lambda \xrightarrow{a \overset{A}{\mapsto} b} M'$ has been derived using the rule for transitions up to α-equivalence given in Definition 14 on page 15. This means that

$[\![M]\!]^\lambda \approx_\alpha N_1 \stackrel{a \stackrel{A}{\mapsto} b}{\longrightarrow} N_2 \approx_\alpha M'$ for some N_1 and N_2, where the proof of $N_1 \stackrel{a \stackrel{A}{\mapsto} b}{\longrightarrow} N_2$ is shorter than the proof of $[\![M]\!]^\lambda \stackrel{a \stackrel{A}{\mapsto} b}{\longrightarrow} M'$. Since $|M| = n$ and we have already shown statement 2_n, we can infer that there is some N_1' such that $M \equiv N_1'$ and $[\![N_1']\!]^\lambda = N_1$. As $|N_1'| = |M| = n$, we may now apply the inner inductive hypothesis to $[\![N_1']\!]^\lambda = N_1 \stackrel{a \stackrel{A}{\mapsto} b}{\longrightarrow} N_2$ to derive that there is some N_2' such that $[\![N_2']\!]^\lambda = N_2$ and $N_2' \equiv N_1'[b/a]$. Observe now that

$$|N_2'| = |N_1'[b/a]| = |N_1'| = |M| = n.$$

Therefore, we may again use statement 2_n to infer that there is some N' such that $N' \equiv N_2'$ and $[\![N']\!]^\lambda = M'$. In summary, as $M \equiv N_1'$, we have that

$$M[b/a] \equiv N_1'[b/a] \equiv N_2' \equiv N',$$

and, using transitivity of $\equiv$, we are done.

Proof of statement 4_n. We prove the claim in the following steps, where M is a λ-term of size n.

1. If $M[b/a]$ is defined (without any need for α-conversion) then $[\![M]\!]^\lambda \stackrel{a \stackrel{A}{\mapsto} b}{\longrightarrow} [\![M[b/a]]\!]^\lambda$.
2. If $M \equiv M'$ and $M'[b/a]$ is defined (without any need for α-conversion) then $[\![M]\!]^\lambda \stackrel{a \stackrel{A}{\mapsto} b}{\longrightarrow} [\![M[b/a]]\!]^\lambda$.
3. If $N \equiv M[b/a]$ then $[\![M]\!]^\lambda \stackrel{a \stackrel{A}{\mapsto} b}{\longrightarrow} [\![N]\!]^\lambda$.

The first item above is shown by an inner induction on the structure of M. The details are similar to those for the proof of statement 3_n and are therefore omitted. To prove the second claim, one uses 1_n (twice), the first item and the rule for transitions up to α-equivalence given in Definition 14 on page 15. The third claim follows using 1_n, the second item and the rule for transitions up to α-equivalence given in Definition 14 on page 15. ⊠

We now prove the two implications in the statement of Lemma 5, namely

- If $[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} M'$ then $M' = [\![M'']\!]^\lambda$ for some $M'' \equiv M[N/a]$.
- If $M[N/a] \equiv M'$ then $[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} [\![M']\!]^\lambda$.

Proof of the statement: If $[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} M'$ then $M' = [\![M'']\!]^\lambda$ for some $M'' \equiv M[N/a]$. We show the claim by induction on the proof of the transition

$$[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} M'.$$

We proceed by a case analysis on the last rule used in the proof and limit ourselves to presenting the details for two selected cases. The other cases are proved in similar fashion.

– Assume that $M = \lambda c.M_1$ for some M_1 and some atom c. Then

$$[\![M]\!]^\lambda = \lambda([c][\![M_1]\!]^\lambda) \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} M'$$

has been shown using rule (λ_{Ts}) in Figure 3 on page 17. This means that, for some M_1',

- $M' = \lambda([c]M_1')$ and
- $[c][\![M_1]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} [c]M_1'$.

If $c = a$ then the above transition has been derived using rule ($\mathrm{abs2}_{\mathrm{Ts}}$). This means that $M' = [\![M]\!]^\lambda$. Since $M = M[N/a]$, we are done.

Assume now that $c \neq a$. In this case, the transition $[c][\![M_1]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} [c]M_1'$ has been derived using rule ($\mathrm{abs1}_{\mathrm{Ts}}$) in Figure 3. This means that $c\#[\![N]\!]^\lambda$ and, by a shorter inference, $[\![M_1]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} M_1'$. By induction, there is some M_1'' such that $M_1'' \equiv M_1[N/a]$ and $[\![M_1'']\!]^\lambda = M_1'$. Let $M'' = \lambda c.M_1''$. Then, as $c \neq a$ and $c \notin FV(N)$ by Lemma 6,

$$M[N/a] = \lambda c.(M_1[N/a]) \equiv \lambda c.M_1'' = M''.$$

Moreover, $[\![M'']\!]^\lambda = M'$ and we are done.

– Assume that $[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} M'$ has been derived using the rule for transitions up to α-equivalence given in Definition 14 on page 15. This means that $[\![M]\!]^\lambda \approx_\alpha N_1 \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} N_2 \approx_\alpha M'$ for some N_1 and N_2, where the proof of $N_1 \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} N_2$ is shorter than the proof of $[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} M'$. By Lemma 4, we can infer that there is some N_1' such that $M \equiv N_1'$ and $[\![N_1']\!]^\lambda = N_1$. We may therefore apply the inductive hypothesis to $[\![N_1']\!]^\lambda = N_1 \stackrel{a \stackrel{T}{\mapsto} [\![N_1']\!]^\lambda}{\longrightarrow} N_2$ to derive that there is some N_2' such that $[\![N_2']\!]^\lambda = N_2$ and $N_2' \equiv N_1'[N/a]$. Now, we may again use Lemma 4 to infer that there is some M'' such that $M'' \equiv N_2'$ and $[\![M'']\!]^\lambda = M'$. In summary, as $M \equiv N_1'$, we have that

$$M[N/a] \equiv N_1'[N/a] \equiv N_2' \equiv M'',$$

and, using transitivity of $\equiv$, we are done.

Proof of the statement: If $M[N/a] \equiv M'$ then $[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} [\![M']\!]^\lambda$. We prove the claim in the following steps, where M and N are λ-terms.

1. If $M[b/a]$ is defined (without any need for α-conversion) then $[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} [\![M[N/a]]\!]^\lambda$.
2. If $M \equiv M'$ and $M'[N/a]$ is defined (without any need for α-conversion) then $[\![M]\!]^\lambda \stackrel{a \stackrel{T}{\mapsto} [\![N]\!]^\lambda}{\longrightarrow} [\![M[N/a]]\!]^\lambda$.

3. If $M' \equiv M[N/a]$ then $[\![M]\!]^\lambda \stackrel{a \mapsto [\![N]\!]^\lambda}{\longrightarrow}_T [\![M']\!]^\lambda$.

The first item above is shown by induction on the structure of M. The details are similar to those for the proof of the previous statement and are therefore omitted. To prove the second claim, one uses Lemma 4, the first item and the rule for transitions up to α-equivalence given in Definition 14 on page 15. The third claim follows using Lemma 4, the second item and the rule for transitions up to α-equivalence given in Definition 14 on page 15.

F Proof of Theorem 5

For ease of reference, we recall that the encoding $[\![\cdot]\!]^\pi$ is the map from Π into terms of the nominal early π-calculus defined in Section 5.2 as follows.

$$\begin{aligned}
[\![\mathbf{0}]\!]^\pi &= \mathbf{0} \\
[\![\tau.P]\!]^\pi &= \tau.[\![P]\!]^\pi \\
[\![\overline{a}b.P]\!]^\pi &= out(a, b, [\![P]\!]^\pi) \\
[\![a(b).P]\!]^\pi &= in(a, [b][\![P]\!]^\pi) \\
[\![\nu a.P]\!]^\pi &= \nu([a][\![P]\!]^\pi) \\
[\![P + Q]\!]^\pi &= [\![P]\!]^\pi + [\![Q]\!]^\pi \\
[\![P \,||\, Q]\!]^\pi &= [\![P]\!]^\pi \,||\, [\![Q]\!]^\pi \\
[\![!P]\!]^\pi &= ![\![P]\!]^\pi
\end{aligned}$$

The encoding is extended to labels thus:

$$\begin{aligned}
[\![\tau]\!]^\pi &= \tau, \\
[\![ab]\!]^\pi &= in(a, b), \\
[\![\overline{a}b]\!]^\pi &= out(a, b) \quad \text{and} \\
[\![\overline{a}(b)]\!]^\pi &= bout(a, [b]).
\end{aligned}$$

Our proof of Theorem 5 relies on the following lemmas, stating the correctness of substitution and α-conversion transitions.

Lemma 9 (Correctness of substitution transitions). *For all $P \in \Pi$, atoms a and b and $P' \in \mathbb{C}(\Sigma^\pi)$, it holds that*

$$[\![P]\!]^\pi \stackrel{a \mapsto b}{\longrightarrow} P' \text{ if and only if } P' = [\![P[b/a]]\!]^\pi.$$

We use $\equiv$ to denote syntactic equality up to α-equivalence of the early π-calculus.

Lemma 10 (Correctness of α-conversion transitions).

- $\Rightarrow$: *For all $P \in \Pi$ and $Q' \in \mathbb{C}(\Sigma^\pi)$, if $[\![P]\!]^\pi \approx_\alpha Q'$ then there exists $Q \in \Pi$ such that $P \equiv Q \wedge [\![Q]\!]^\pi = Q'$.*
- $\Leftarrow$: *For all $P, Q \in \Pi$, if $P \equiv Q$ then $[\![P]\!]^\pi \approx_\alpha [\![Q]\!]^\pi$.*

The proofs of Lemmas 9 and 10 are lengthy even though they follow standard reasoning by induction. They can be found in Section G and Section H, respectively.

For ease of reference, we repeat below the rules for the early π-calculus given in [48][10]. In the rules that follow, $names(\alpha)$ denotes the set of names that occur in the label α, $bn(\alpha)$ denotes the set of bound names that occur in the label α and $fn(P)$ denotes the set of names of the process P that are not bound, see [48]. For completeness of reference, the definition of these sets are repeated in Section C. Although these notions are standard, Section C also repeats the definitions of substitution and α-equivalence of π-calculus.

$$\frac{}{\tau.x \xrightarrow{\tau} x}\ (\tau\text{O}) \qquad \frac{}{\overline{a}b.x \xrightarrow{\overline{a}b} x}\ (\text{outO}) \qquad \frac{}{a(b).x \xrightarrow{ac} y[c/b]}\ (\text{inO})$$

$$\frac{x_1 \xrightarrow{\alpha} y_1}{x_1 + x_2 \xrightarrow{\alpha} y_1}\ (\text{sumO1}) \qquad bn(\alpha) \cap fn(x_2) = \emptyset\ \frac{x_1 \xrightarrow{\alpha} y_1}{x_1 \,||\, x_2 \xrightarrow{\alpha} y_1 \,||\, x_2}\ (\text{parO1})$$

$$\frac{x_1 \xrightarrow{\overline{a}b} y_1 \quad x_2 \xrightarrow{ab} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau} y_1 \,||\, y_2}\ (\text{comO1})$$

$$b \notin fn(x_2)\ \frac{x_1 \xrightarrow{\overline{a}(b)} y_1 \quad x_2 \xrightarrow{ab} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau} \nu b.(y_1 \,||\, y_2)}\ (\text{closeO1}) \qquad \frac{x \xrightarrow{\alpha} y}{!x \xrightarrow{\alpha} y \,||!x}\ (\text{replO})$$

$$a \neq z\ \frac{x \xrightarrow{\overline{z}a} y}{\nu a.x \xrightarrow{\overline{z}(a)} y}\ (\text{openO}) \qquad c \notin names(\alpha)\ \ \frac{x \xrightarrow{\alpha} y}{\nu c.x \xrightarrow{\alpha} \nu c.x}\ (\text{resO})$$

For the sake of brevity, we omit the symmetric versions of rules (sumO1), (parO1), (parResO1), (comO1) and (closeO1). In what follows, these are referred to as (sumO2), (parO2), (parResO2), (comO2) and (closeO2).

The proof of Theorem 5 is divided into two cases

- $\Rightarrow$: if $P \xrightarrow{\alpha} Q$ then $[\![P]\!]^\pi \xrightarrow{[\![\alpha]\!]^\pi} [\![Q]\!]^\pi$
- $\Leftarrow$: if $[\![P]\!]^\pi \xrightarrow{\alpha} Q$ then $P \xrightarrow{\alpha'} Q'$ for some α' and Q' such that $[\![\alpha']\!]^\pi = \alpha$ and $[\![Q']\!]^\pi = Q$

We recall that in the statement for $\Leftarrow$ the label α does not range over substitution and α-conversion transition labels.

The following lemmas will be necessary to complete the proof. Their proofs are straightforward and thus omitted.

[10] [48] presents more rules for the replication operator for technical reasons we are not concerned with, see pages 42 and 43 in that reference.

Lemma 11 (Interplay between the set of names and freshness (1).). *Given a channel name a and a π-calculus label α, it holds that if $a \notin names(\alpha)$ then $a \notin bn(\llbracket\alpha\rrbracket^\pi)$ and the freshness assertion $a\#\llbracket\alpha\rrbracket^\pi$ is derivable.*

Lemma 12 (Interplay between the set of names and freshness (2).). *Given a channel name a and a π-calculus label α. Let α' be a label such that $\alpha' = \llbracket\alpha\rrbracket^\pi$. It holds that if $a \notin bn(\llbracket\alpha\rrbracket^\pi)$ and $a\#\llbracket\alpha\rrbracket^\pi$, then $a \notin names(\alpha)$.*

Lemma 13 (Freshness in terms and their encodings.). *Given a channel name a and a π-calculus process P. a is fresh in $\llbracket P\rrbracket^\pi$ if and only if $a \notin fn(P)$.*

$\Rightarrow$

The proof is by induction on the derivation of the π-calculus transition. We proceed by a case analysis on the last rule used in the derivation.

- $\tau.P$: The only possible transition is $\tau.P \xrightarrow{\tau} P$. Now $\llbracket\tau.P\rrbracket^\pi = \tau.\llbracket P\rrbracket^\pi$, and by rule ($\tau$) we can prove the transition $\tau.\llbracket P\rrbracket^\pi \xrightarrow{\tau} \llbracket P\rrbracket^\pi$.
- $\overline{a}b.P$: The only possible transition is $\overline{a}b.P \xrightarrow{\overline{a}b} P$. Now, $\llbracket\overline{a}b.P\rrbracket^\pi = out(a, b, \llbracket P\rrbracket^\pi)$, and by rule (out), we can prove the transition $out(a, b, \llbracket P\rrbracket^\pi) \overset{out(a,b)}{\rightarrow} \llbracket P\rrbracket^\pi$.
- $a(b).P$: The possible transitions are with labels ac for all atoms c. Let us pick an atom c and consider a transition $a(b).P \xrightarrow{ac} P[c/b]$. Now, $\llbracket a(b).P\rrbracket^\pi = in(a, [b]\llbracket P\rrbracket^\pi)$. By Lemma 9, which states the correctness of substitution transitions, we know that $\llbracket P\rrbracket^\pi \xrightarrow{a\mapsto c} \llbracket P[c/a]\rrbracket^\pi$, thus we can use rule (in) to prove the transition $in(a, [b]\llbracket P\rrbracket^\pi) \overset{in(a,c)}{\rightarrow} \llbracket P[c/a]\rrbracket^\pi$.
- $\nu b.P$: Let us recall the fact that $\llbracket\nu b.P\rrbracket^\pi = \nu([b]\llbracket P\rrbracket^\pi)$. There are two possible transitions.
  - $\nu b.P \xrightarrow{\overline{a}(b)} P'$ by rule (openO), with $P \xrightarrow{\overline{a}b} P'$ and $b \neq a$. By the inductive hypothesis $\llbracket P\rrbracket^\pi \overset{out(a,b)}{\rightarrow} \llbracket P'\rrbracket^\pi$, thus the premise in rule (open) is satisfied, and we can use this rule to prove a transition $\nu([b]\llbracket P\rrbracket^\pi) \overset{bout(a,[b])}{\rightarrow} \llbracket P'\rrbracket^\pi$.
  - $\nu b.P \xrightarrow{\alpha} \nu b.P'$ by rule (resO), with $P \xrightarrow{\alpha} P'$ and for a generic α such that $b \notin names(\alpha)$. By inductive hypothesis $\llbracket P\rrbracket^\pi \xrightarrow{\alpha} \llbracket P'\rrbracket^\pi$. By Lemma 11, since $b \notin names(\alpha)$ we have that $b \notin bn(\llbracket\alpha\rrbracket^\pi)$ and $b\#\llbracket\alpha\rrbracket^\pi$, thus the premises in rule (res) are satisfied, and we can instanciate this rule in the following way

    $$b \notin bn(\alpha) \quad \frac{\llbracket P\rrbracket^\pi \xrightarrow{\alpha} \llbracket P'\rrbracket^\pi \quad b\#\alpha}{\nu([b]\llbracket P\rrbracket^\pi) \xrightarrow{\alpha} \nu([b]\llbracket P'\rrbracket^\pi)}\ (\text{res})$$

    to prove a transition $\nu([b]\llbracket P\rrbracket^\pi) \xrightarrow{\alpha} \nu([b]\llbracket P'\rrbracket^\pi)$, whose target is exactly $\llbracket\nu b.P'\rrbracket^\pi$.
- $P + Q$: Consider a generic transition α. There are two possible transitions, (1) $P + Q \xrightarrow{\alpha} P'$, with $P \xrightarrow{\alpha} P'$ and (2) $P + Q \xrightarrow{\alpha} Q'$, with $Q \xrightarrow{\alpha} Q'$. We here consider only (1). Now, $\llbracket P+Q\rrbracket^\pi = \llbracket P\rrbracket^\pi + \llbracket Q\rrbracket^\pi$. Since $P \xrightarrow{\alpha} P'$, by inductive hypothesis $\llbracket P\rrbracket^\pi \xrightarrow{\alpha} \llbracket P'\rrbracket^\pi$, thus the premise in rule (sum1) is satisfied, and

we can use this rule to prove a transition $[\![P]\!]^\pi + [\![Q]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi$. Case (2) can be proven analogously.

- $P \,||\, Q$: Then $[\![P \,||\, Q]\!]^\pi = [\![P]\!]^\pi \,||[\![Q]\!]^\pi$. We distinguish four cases.
  - $\alpha \notin \{bout(a,[b]) \mid a,b \in C\}$: There are two possible transitions, (1) $P \,||\, Q \xrightarrow{\alpha} P' \,||\, Q$, with $P \xrightarrow{\alpha} P'$ and (2) $P \,||\, Q \xrightarrow{\alpha} P \,||\, Q'$, with $Q \xrightarrow{\alpha} Q'$. (We treat below separately the case for $\alpha = \tau$ with a top level communication taking place). We here consider only (1). Now, $[\![P \,||\, Q]\!]^\pi = [\![P]\!]^\pi \,||[\![Q]\!]^\pi$. Since $P \xrightarrow{\alpha} P'$, by inductive hypothesis $[\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi$, thus the premise in rule (par1) is satisfied, and we can use this rule to prove a transition $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi \,||[\![Q]\!]^\pi$, whose target is exactly $[\![P' \,||\, Q]\!]^\pi$. Case (2) can be proven analogously.
  - $\alpha \in \{bout(a,[b]) \mid a,b \in C\}$: The only possible transition is, by rule (parO1), $P \,||\, Q \xrightarrow{bout(a,[b])} P' \,||\, Q$, with $P \xrightarrow{bout(a,[b])} P'$ and $b \notin fn(Q)$. Since $P \xrightarrow{\alpha} P'$, by inductive hypothesis $[\![P]\!]^\pi \xrightarrow{bout(a,[b])} [\![P']\!]^\pi$. Moreover, since $b \notin fn(Q)$, by Lemma 13 we have that $b\#[\![Q]\!]^\pi$. The two premises of rules (par1) are therefore satisfied, and we can use this rule to prove a transition$[\![P]\!]^\pi \,||[\![Q]\!]^\pi \xrightarrow{bout(a,[b])} [\![P']\!]^\pi \,||[\![Q]\!]^\pi$, whose target is exactly $[\![P' \,||\, Q]\!]^\pi$.
  - Assume that $P \,||\, Q \xrightarrow{\tau} P' \,||\, Q'$ by rule (comO1), with $P \xrightarrow{\overline{a}b} P'$ and $Q \xrightarrow{ab} Q'$ for some atoms a and b. Since $P \xrightarrow{\overline{a}b} P'$ and $Q \xrightarrow{ab} Q'$, by the inductive hypothesis $[\![P]\!]^\pi \xrightarrow{out(a,b)} [\![P']\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{in(a,b)} [\![Q']\!]^\pi$, thus the premises in rule (com1) are satisfied, and we can use this rule to prove a transition $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \xrightarrow{\tau} [\![P']\!]^\pi \,||[\![Q']\!]^\pi$, whose target is exactly $[\![P' \,||\, Q']\!]^\pi$.
  - Assume that $P \,||\, Q \xrightarrow{\tau} \nu b.(P' \,||\, Q')$ by rule (closeO1), with $P \xrightarrow{\overline{a}(b)} P'$ and $Q \xrightarrow{ab} Q'$ for some atoms a and b, with $b \notin fn(Q)$. Now, since $P \xrightarrow{\overline{a}(b)} P'$ and $Q \xrightarrow{ab} Q'$, by the inductive hypothesis $[\![P]\!]^\pi \xrightarrow{bout(a,[b])} [\![P']\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{in(a)(b)} [\![Q']\!]^\pi$. The premises of rule (close1) are all satisfied and we can use this rule to prove a transition $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \xrightarrow{\tau} \nu([b]([\![P']\!]^\pi \,||[\![Q']\!]^\pi))$, whose target is exactly $[\![\nu b.(P' \,||\, Q')]\!]^\pi$.
- $!P$: Assume that $!P \xrightarrow{\alpha} P' \,||\, !P$, with $P \xrightarrow{\alpha} P'$. Now, $[\![!P]\!]^\pi = !\,[\![P]\!]^\pi$. Since $P \xrightarrow{\alpha} P'$, by the inductive hypothesis $[\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi$. Thus the premise in rule (repl) is satisfied, and we can use this rule to prove a transition $![\![P]\!]^\pi \xrightarrow{\alpha} [\![P']\!]^\pi \,||\, ![\![P]\!]^\pi$, whose target is exactly $[\![P' \,||\, !P]\!]^\pi$.

$\Leftarrow$

The proof is by induction on the structure of the π-term P and then by cases on the last rule used in the derivation of the transition $[\![P]\!]^\pi \xrightarrow{\alpha} Q$.

- $\mathbf{0}$: $[\![\mathbf{0}]\!]^\pi = \mathbf{0}$. Since α does not range over substitution and α-conversion labels, this case is vacuous.
- $\tau.P$: $[\![\tau.P]\!]^\pi = \tau.[\![P]\!]^\pi$. The only transition is $\tau.[\![P]\!]^\pi \xrightarrow{\tau} [\![P]\!]^\pi$ by rule (τ). In this case, $\tau.P \xrightarrow{\tau} P$ with $[\![\tau]\!]^\pi = \tau$, and we are done.

- $\overline{a}b.P$: $[\![\overline{a}b.P]\!]^\pi = out(a, b, [\![P]\!]^\pi)$. The only transition is $out(a, b, [\![P]\!]^\pi) \overset{out(a,b)}{\to} [\![P]\!]^\pi$, by rule (out). In this case, $\overline{a}b.P \overset{\overline{a}b}{\to} P$ with $[\![\overline{a}b]\!]^\pi = out(a, b)$ and we are done.
- $a(b).P$: $[\![a(b).P]\!]^\pi = in(a, [b][\![P]\!]^\pi)$. The only possible transitions have labels of the form $in(a, c)$. Let us pick an action $in(a, c)$. Then $in(a, [b][\![P]\!]^\pi) \overset{in(a,c)}{\to} P'$, by rule (in), with $[\![P]\!]^\pi \overset{a\mapsto c}{\longrightarrow} P'$. By Lemma 9, which states the correctness of substitution transitions, $P' = [\![P[c/a]]\!]^\pi$. Since $[\![ac]\!]^\pi = in(a, c)$ and $\overline{a}b.P \overset{ac}{\to} P[c/a]$ we are done.
- $\nu b.P$: $[\![\nu a.P]\!]^\pi = \nu([a][\![P]\!]^\pi)$. There are two possible transitions.
  - $\nu([b][\![P]\!]^\pi) \overset{bout(a,[b])}{\to} P'$ by rule (open), with $[\![P]\!]^\pi \overset{out(a,b)}{\to} P'$ and $a \neq b$. By the inductive hypothesis $[\![P']\!]^\pi = P''$ and $P \overset{\overline{a}b}{\to} P''$ for some P''. We can now use the rule (open) in order to prove a transition $\nu b.P \overset{\overline{a}(b)}{\to} P''$ with $[\![\overline{a}(b)]\!]^\pi = bout(a, [b])$ and we are done.
  - $\nu([b][\![P]\!]^\pi) \overset{\alpha}{\to} \nu([b]P')$by rule (res), with $[\![P]\!]^\pi \overset{\alpha}{\to} P'$ and $b \notin bn(\alpha)$ and $b\#\alpha$. By inductive hypothesis $P \overset{\alpha'}{\to} P''$ for some P'' such that $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. By Lemma 12, since $b \notin bn(\alpha)$ and $b\#\alpha$ then we have that $b \notin names(\alpha')$. We can thus use the rule (resO) in order to prove a transition $\nu b.P \overset{\alpha'}{\to} \nu b.P''$ with $\alpha = [\![\alpha']\!]^\pi$ and we are done.
- $P + Q$: $[\![P + Q]\!]^\pi = [\![P]\!]^\pi + [\![Q]\!]^\pi$. There are two possible transitions, (1) $[\![P]\!]^\pi + [\![Q]\!]^\pi \overset{\alpha}{\to} P'$, by rule (sum1), with $[\![P]\!]^\pi \overset{\alpha}{\to} P'$, or (2) $[\![P]\!]^\pi + [\![Q]\!]^\pi \overset{\alpha}{\to} Q'$, by rule (sum2), with $[\![Q]\!]^\pi \overset{\alpha}{\to} Q'$. Here we consider only transition (1). By inductive hypothesis $P \overset{\alpha'}{\to} P''$ for some P'' such that $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. We can thus prove a transition $P + Q \overset{\alpha'}{\to} P''$, with $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. Case (2) can be proven analogously.
- $P \,||\, Q$: $[\![P \,||\, Q]\!]^\pi = [\![P]\!]^\pi \,||[\![Q]\!]^\pi$. We distinguishes four cases:
  - $\alpha \notin \{\overline{a}(b) \mid a, b \in C\}$: There are two possible transitions, (1) $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \overset{\alpha}{\to} P' \,||\, Q$, by rule (par1), with $[\![P]\!]^\pi \overset{\alpha}{\to} P'$, or (2) $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \overset{\alpha}{\to} P \,||\, Q'$, by rule (par2), with $[\![Q]\!]^\pi \overset{\alpha}{\to} Q'$. (We treat below separately the case for $\alpha = \tau$ with a top level communication taking place). Here we consider only transition (1). By inductive hypothesis $P \overset{\alpha'}{\to} P''$ for some P'' such that $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. We can thus prove a transition $P \,||\, Q \overset{\alpha'}{\to} P'' \,||\, Q$, whose target is exactly $[\![P' \,||\, Q]\!]^\pi$ and with $\alpha = [\![\alpha']\!]^\pi$. Case (2) can be proven analogously.
  - $\alpha = \overline{a}(b)$: There are two possible transitions: (1) $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \overset{\alpha}{\to} P' \,||\, Q$, by rule (parRes1), with $[\![P]\!]^\pi \overset{bout(a,[b])}{\to} P'$ and b fresh in $[\![Q]\!]^\pi$, or (2) $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \overset{bout(a,[b])}{\to} P \,||\, Q'$, by rule (parRes2), with $[\![Q]\!]^\pi \overset{bout(a,[b])}{\to} Q'$ and b fresh in $[\![P]\!]^\pi$, Here we consider only transition (1). By the inductive hypothesis, $P' = [\![P'']\!]^\pi$ and $P \overset{\overline{a}(b)}{\to} P''$ for some P''. By Lemma 13, since b is fresh in $[\![Q]\!]^\pi$ we have that $b \notin fn(Q)$. We can thus prove a transition $P \,||\, Q \overset{\overline{a}(b)}{\to} P'' \,||\, Q$, whose target is exactly $[\![P' \,||\, Q]\!]^\pi$, and with $[\![\overline{a}(b)]\!]^\pi = bout(a, [b])$.

- $\alpha = \tau$ using (com1): $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \xrightarrow{\tau} P' \,||\, Q'$, with $[\![P]\!]^\pi \xrightarrow{out(a,b)} P'$ and $[\![Q]\!]^\pi \xrightarrow{in(a,b)} Q'$. By the inductive hypothesis, $P' = [\![P'']\!]^\pi$and $P \xrightarrow{\overline{a}b} P''$ for some P'', and also $[\![Q']\!]^\pi = Q''$ and $Q \xrightarrow{ab} Q''$ for some Q''. We can thus use the rule (comO1) in order to prove a transition $P \,||\, Q \xrightarrow{\tau} P'' \,||\, Q''$, whose target is exactly $[\![P' \,||\, Q']\!]^\pi$, and with $[\![\tau]\!]^\pi = \tau$. When using the rule (com2) the same reasoning applies.
- $\alpha = \tau$ using (close1): $[\![P]\!]^\pi \,||[\![Q]\!]^\pi \xrightarrow{\tau} \nu([b](P' \,||\, Q'))$, with $[\![P]\!]^\pi \xrightarrow{bout(a,[b])} P'$, $[\![Q]\!]^\pi \xrightarrow{in(a,b)} Q'$ and b fresh in both $[\![P]\!]^\pi$ and $[\![Q]\!]^\pi$. By the inductive hypothesis, $P' = [\![P'']\!]^\pi$ and $P \xrightarrow{\overline{a}(b)} P''$ for some P'', and also $[\![Q']\!]^\pi = Q''$ and $Q \xrightarrow{ab} Q''$ for some Q''. By Lemma 13, since b is fresh in $[\![Q]\!]^\pi$ we have that $b \notin fn(Q)$. We can thus use the rule (closeO1) in order to prove a transition $P \,||\, Q \xrightarrow{\tau} \nu b.(P'' \,||\, Q'')$, whose target is exactly $[\![\nu([b](P' \,||\, Q'))]\!]^\pi$, and with $[\![\tau]\!]^\pi = \tau$. When using the rule (close2) the same reasoning applies.

– $!P$: $[\![!P]\!]^\pi = !\,[\![P]\!]^\pi$. The only transition is $!\,[\![P]\!]^\pi \xrightarrow{\alpha} P' \,||!\,[\![P]\!]^\pi$, by rule (repl), with $[\![P]\!]^\pi \xrightarrow{\alpha} P'$. By inductive hypothesis $P \xrightarrow{\alpha'} P''$ for some P'' such that $P' = [\![P'']\!]^\pi$ and $\alpha = [\![\alpha']\!]^\pi$. We can thus prove a transition $!P \xrightarrow{\alpha'} P'' \,||!P$, whose target is exactly $[\![P' \,||!P]\!]^\pi$, and with $\alpha = [\![\alpha']\!]^\pi$.

G Correctness of substitutions for π: Proof of Lemma 9

The proof of Lemma 9 is divided into two cases.

– $\Rightarrow$: $[\![P]\!]^\pi \xrightarrow{a \overset{A}{\mapsto} b} [\![P[b/a]]\!]^\pi$

– $\Leftarrow$: if $P[b/a] = P'$ then $[\![P]\!]^\pi \xrightarrow{a \overset{A}{\mapsto} b} [\![P']\!]^\pi$

The reader must be aware of the fact that atom-for-atom substitutions are up to α-equivalence and simultaneously that α-conversion transitions are defined upon atom-for-atom substitutions. Proving Lemma 9 and Lemma 10, which states the correctness of α-conversion transitions, separately results in a mutual reference. The mutual reference is however harmless and we can proceed by proving the two lemmas together in a big induction on the sum of the lengths of the proofs for substitution transitions and α-conversion transitions.

We however prefer to keep the main concepts of the proofs separate and present the proof of Lemma 9 in a standalone fashion, as though we can safely refer to Lemma 10. The reader should however always keep in mind how the actual proof proceeds and that this is just for presentation sake.

$\Rightarrow$

Let us pick closed a π-term P and atoms a and b. The proof is by induction on the length of the proof of transition $\xrightarrow{a \overset{A}{\mapsto} b}$ from the encoding of the π-term P.

– Proofs of length 1: The only proof of length 1 is with rule $(\mathbf{0}_{\mathrm{s}})$ from the encoding of the term $\mathbf{0}$. In fact, since the constant $\mathbf{0}$ has no parameters, the rule $(\mathbf{0}_{\mathrm{s}})$ turns out to be an axiom. We have that $\mathbf{0}[b/a] = \mathbf{0}$ and since $[\![\mathbf{0}]\!]^{\pi} = \mathbf{0}$ we can prove $[\![\mathbf{0}]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} [\![\mathbf{0}]\!]^{\pi}$ by rule $(\mathbf{0}_{\mathrm{s}})$.
– Proofs of length $n > 1$: There are different proofs of length $n > 1$ to consider. In this proof we limit ourselves to show the cases for the π-terms of the form $\tau.P$, $P \,||\, Q$ and $c(d).P$. The proof cases for the other forms can be subsumed by those. The case for the application of an α-conversion with the rule $\xrightarrow{a \mapsto^{A} b}$ ·upToα is also considered.
  - (τ_{s}) from $[\![\tau.P]\!]^{\pi}$: We have that $(\tau.P)[b/a] = \tau.(P[b/a])$. We can instantiate the rule (τ_{s}) as below.

$$\frac{[\![P]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} P'}{\tau.[\![P]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} \tau.P'}$$

    Since the length of the proof for the transitions $[\![P]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} P'$ is strictly less than the length of the proof for $\tau.[\![P]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} \tau.P'$, the inductive hypothesis applies and we can conclude that $P' = [\![P[b/a]]\!]^{\pi}$. We therefore have that $\tau.[\![P]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} \tau.[\![P[b/a]]\!]^{\pi}$, that is $[\![\tau.P[b/a]]\!]^{\pi}$.
  - $(||_{\mathrm{s}})$ from $[\![P \,||\, Q]\!]^{\pi}$: We have that $(P \,||\, Q)[b/a] = (P[b/a] \,||\, Q[b/a])$. We can instantiate the rule $(||_{\mathrm{s}})$ as below.

$$\frac{[\![P]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} P' \qquad [\![Q]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} Q'}{\tau.[\![P \,||\, Q]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} P' \,||\, Q'}$$

    Since the length of the proofs for the transitions $[\![P]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} P'$ and $[\![Q]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} Q'$ are strictly less than the length of the proof for $[\![P \,||\, Q]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} P' \,||\, Q'$, the inductive hypothesis applies and we can conclude that $P' = [\![P[b/a]]\!]^{\pi}$ and $Q' = [\![Q[b/a]]\!]^{\pi}$. We therefore have that $[\![P \,||\, Q]\!]^{\pi} \xrightarrow{a \mapsto^{A} b} [\![P[b/a]]\!]^{\pi} \,||\, [\![Q[b/a]]\!]^{\pi}$, that is $[\![P[b/a] \,||\, Q[b/a]]\!]^{\pi}$.
  - $(\mathrm{in}_{\mathrm{s}})$ from $[\![c(d).P]\!]^{\pi}$:
    - substitution $(c(d).P)[a/d]$: The atom c might be equal to d. We have that $(c(d).P)[a/d] = c[d/a](d).P$. Recalling that $[\![c(d).P]\!]^{\pi} = in(c, [d].[\![P]\!]^{\pi})$, we can instantiate the rule $(\mathrm{in}_{\mathrm{s}})$ as below.

$$\frac{c \xrightarrow{a \mapsto^{A} d} c[d/a] \qquad [d].[\![P]\!]^{\pi} \xrightarrow{a \mapsto^{A} d} [d].[\![P]\!]^{\pi}}{in(c, [d].[\![P]\!]^{\pi}) \xrightarrow{a \mapsto^{A} d} in(c[d/a], [d].[\![P]\!]^{\pi})}$$

Where the transition $d].[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}d} [d].[\![P]\!]^\pi$ is provable by means of the axiom $(abs2_{\mathrm{As}})$. We therefore have that $in(c,[d].[\![P]\!]^\pi) \xrightarrow{a\overset{A}{\mapsto}d} in(c[d/a],[d].[\![P]\!]^\pi)$, that is $[\![(c(d).P)[a/d]]\!]^\pi$.

* substitution $(c(d).P)[b/a]$, with $a \neq d$ and b fresh in P: The atom c might be equal to a. We have that $(c(d).P)[b/a] = c[[b/a](d).(P[b/a])$. Before instantiating the rule $(\mathrm{in_s})$ we can see how we can instantiate the rule $(abs1_{\mathrm{Ts}})$ to suit our purposes. Let us consider the following instantiation of $(abs1_{\mathrm{Ts}})$.

$$\frac{[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} P' \quad b\#[\![P]\!]^\pi \quad a \neq d}{[d].[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} [d].P'}$$

By Lemma 13, since b is fresh in P then b is fresh also in $[\![P]\!]^\pi$ and $b\#[\![P]\!]^\pi$ holds. Since the length of the proofs for the transition $[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} P'$ is strictly less than the length of the proof for $[d].[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} [d].P'$, the inductive hypothesis applies and we can conclude that $P' = [\![P[b/a]]\!]^\pi$. The transition proved above is thus $[d].[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} [d].[\![P[b/a]]\!]^\pi$. Now we can instantiate the rule$(\mathrm{in_s})$ as below.

$$\frac{c \xrightarrow{a\overset{A}{\mapsto}b} c[b/a] \qquad [d].[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} [d].[\![P[b/a]]\!]^\pi}{in(c,[d].[\![P]\!]^\pi) \xrightarrow{a\overset{A}{\mapsto}b} in(c[b/a],[d].[\![P[b/a])]\!]^\pi}$$

And we therefore prove $[\![c(d).P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} [\![(c(d).P)[b/a]]\!]^\pi$.

- $\xrightarrow{a\overset{A}{\mapsto}b}$ ·upToα from $[\![P]\!]^\pi$: We can instantiate the rule $\xrightarrow{a\overset{A}{\mapsto}b}$ ·upToα as below.

$$\frac{[\![P]\!]^\pi \approx_\alpha P_1 \quad P_1 \xrightarrow{a\overset{A}{\mapsto}b} P_2}{[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} P_2}$$

Thanks to Lemma 10, which states the correctness of α-conversion transitions, we know that P_1 is such that $P_1 = [\![P_1']\!]^\pi$ for some $P_1' \equiv P$. Since in π-calculus terms are representative of the equivalence class induced by α-equivalence, we have that $P_1 = [\![P]\!]^\pi$. Now, since the length of the proof for the transitions $P_1 \xrightarrow{a\overset{A}{\mapsto}b} P_2$ is strictly less than the length of the proof for the transition $[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} P_2$ (the leave of the tree above), the inductive hypothesis applies and we can conclude that $P_2 = [\![(P[b/a]]\!]^\pi$. We therefore have that$[\![P]\!]^\pi \xrightarrow{a\overset{A}{\mapsto}b} [\![(P[b/a]]\!]^\pi$

$\Leftarrow$

The case $\Leftarrow$ can be proved following the lines of the proof above.

H Correctness of α-conversions for π: Proof of Lemma 10

The proof of Lemma 10 is divided into two cases.

- $\Rightarrow$: if $[\![P]\!]^\pi \approx_\alpha Q'$ then $P \equiv Q \wedge [\![Q]\!]^\pi = Q'$.
- $\Leftarrow$: if $P \equiv Q$ then $[\![P]\!]^\pi \approx_\alpha [\![Q]\!]^\pi$.

Where $\equiv$ is the syntactic equality up to α-equivalence of the early π-calculus.

The reader must be aware of the fact that atom-for-atom substitutions are up to α-equivalence and simultaneously that α-conversion transitions are defined upon atom-for-atom substitutions. Proving Lemma 9, which states the correctness of those substitutions transitions, and Lemma 10 separately results in a mutual reference. The mutual reference is however harmless and we can proceed by proving the two lemmas together in a big induction on the sum of the lengths of the proofs for substitution transitions and α-conversion transitions.

We however prefer to keep the main concepts of the proofs separate and present the proof of Lemma 10 in a standalone fashion, as though we can safely refer to Lemma 9. The reader should however always keep in mind how the actual proof proceeds and that this is just for presentation sake.

$\Rightarrow$

Assuming the hypothesis, let T be a $NTSS$. The proof is by induction on the length of the proofs for α-conversion transitions.

- Proofs of length 1: The only provable transition with length 1 is by rule (id_α), $[\![P]\!]^\pi \approx_\alpha [\![P]\!]^\pi$. Indeed, by reflexivity of $\equiv$, we have that $P \equiv P$.
- Proofs of length n: The rest of the proof proceed in as much the same way as in Lemma 4. In what follows we thus show the proof only for one binder of the ordinary π, namely the restriction $\nu a.P$, and one ordinary operator, namely the parallel operator $||$. The proofs regarding the other binder $a(b).P$, the other operators and the transitivity case are easy to carry out following the same line employed in detail in the proof of Lemma 4.
  - $[\![\nu a.P]\!]^\pi$: There are two possible provable α-conversion transitions for $[\![\nu a.P]\!]^\pi$.
    - $[\![\nu a.P]\!]^\pi \approx_\alpha \nu([b]P')$, with b fresh in $[\![P]\!]^\pi$: Here we first prove $[a][\![P]\!]^\pi \approx_\alpha [b]P'$ using rule $(\mathrm{abs1}_\alpha)$ and then $\nu([a][\![P]\!]^\pi) \approx_\alpha \nu([b]P')$ using the rule (ν_α). Rule $(\mathrm{abs1}_\alpha)$ can be instanciated in the following way

$$\frac{[\![P]\!]^\pi \xrightarrow{a \overset{A}{\rightarrowtail} b} P' \qquad b \# [\![P]\!]^\pi}{[a][\![P]\!]^\pi \approx_\alpha [b]P'}$$

      By Lemma 9, which states the correctness of substitution transitions, we know that $P' = [\![P[b/a]]\!]^\pi$. The transition actually proved by the

rule above is thus $[a][\![P]\!]^\pi \approx_\alpha [b][\![P[b/a]]\!]^\pi$. Now, we can instanciate the rule (ν_α) as follows

$$\frac{[a][\![P]\!]^\pi \approx_\alpha [b][\![P[b/a]]\!]^\pi}{\nu([a][\![P]\!]^\pi) \approx_\alpha \nu([b][\![P[b/a]]\!]^\pi)}$$

in order to prove $\nu([a][\![P]\!]^\pi) \approx_\alpha \nu([b][\![P[b/a]]\!]^\pi)$. The statement of the theorem holds in this case. Indeed, $\nu a.P \equiv \nu b.P[b/a]$.

* $[\![\nu a.P]\!]^\pi \approx_\alpha \nu([a]P')$: Here we address the case where we first prove $[a][\![P]\!]^\pi \approx_\alpha [a]P'$ using rule (abs2_α) and then $\nu([a][\![P]\!]^\pi) \approx_\alpha \nu([a]P')$ using the rule (ν_α). Rule (abs2_α) can be instanciated in the following way

$$\frac{[\![P]\!]^\pi \approx_\alpha P'}{[a][\![P]\!]^\pi \approx_\alpha [a]P'}$$

Since the proof length for proving this transition is strictly less then n, also the proof length for proving $[\![P]\!]^\pi \approx_\alpha P'$ is strictly less then n. By inductive hypothesis we can conclude that $P \equiv P''$, with $[\![P'']\!]^\pi = P'$. Now, we can instanciate the rule (ν_α) as follows

$$\frac{[a][\![P]\!]^\pi \approx_\alpha [a][\![P'']\!]^\pi}{\nu([a][\![P]\!]^\pi) \approx_\alpha \nu([a][\![P'']\!]^\pi}$$

in order to prove $\nu([a][\![P]\!]^\pi) \approx_\alpha \nu([a][\![P'']\!]^\pi$. The statement of the theorem holds in this case. Indeed, since $\equiv$ is a congruence, we can place equated terms in the same context, and since $P \equiv P''$ we have that $\nu a.P \equiv \nu a.P''$.

- $[\![P_1]\!]^\pi \,||[\![P_2]\!]^\pi \approx_\alpha P_1' \,||\, P_2'$ using rule $(||_\alpha)$, with $[\![P_1]\!]^\pi \approx_\alpha P_1'$ and $[\![P_2]\!]^\pi \approx_\alpha P_2'$: Since the proof length for proving these two mentioned transitions is strictly less than n, by inductive hypothesis we can conclude that $P_1 \equiv P_1''$ and $P_2 \equiv P_2''$, with $[\![P_1'']\!]^\pi = P_1'$ and $[\![P_2'']\!]^\pi = P_2'$. As in the previous case, since $\equiv$ is a congruence, $P_1 \,||\, P_2 \equiv P_1'' \,||\, P_2''$, with exactly $[\![P_1'' \,||\, P_2'']\!]^\pi = P_1' \,||\, P_2'$.

$\Leftarrow$

The case $\Leftarrow$ is proved along the line of the case $\Leftarrow$ in proof of Theorem 3. Basically, the change of the bound variable in binders is simulated by rule (abs1_α), the reflexivity is given by rule (id_α), the symmetry is inferred, and the transitivity is given by the rule $\alpha \cdot \text{upTo}\alpha$. Moreover, the reader can see that the rules are such to share α-conversion transitions in any context, which simulates the requirement for $\equiv$ to be a congruence and not just an equivalence relation.

I Bisimilarity when ignoring substitution transitions: Proof of Theorem 6

The proof is divided into two cases, given P and $Q \in \Pi$:

1. Soundness: if $P \leftrightarrows Q$ then $[\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi$.
2. Completeness: if $[\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi$ then $P \leftrightarrows Q$.

Soundness: It suffices to show that the relation

$$\mathcal{R} = \{([\![P]\!]^\pi, [\![Q]\!]^\pi) \mid P \leftrightarrows Q\}$$

satisfies the requirement for $\underline{\leftrightarrow}^-$, i.e., the ones in the definition of the nominal bisimilarity (Definition 16) when we omit to consider substitution transitions.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $[\![P]\!]^\pi \mathcal{R} [\![Q]\!]^\pi$ and $[\![P]\!]^\pi \xrightarrow{\alpha} P'$. In our formulation of the early π-calculus, the action α' may be either (1) an ordinary action, (2) a substitution transition, or (3) an α-conversion transition. Since $\underline{\leftrightarrow}^-$ omits to match substitution transitions, we only need to tackle transitions of form (1) and (3).

Let us consider now the case (1). By Theorem 5, which states the operational correctness of our formulation of π-calculus with respect to the original one, we know that $P \xrightarrow{\alpha'} P''$ for some P'' and α such that $[\![P'']\!]^\pi = P'$ and $[\![\alpha']\!]^\pi = \alpha$. Since $P \leftrightarrows Q$, we have that $Q \xrightarrow{\alpha'} Q''$, with $P'' \leftrightarrows Q''$. By, again, Theorem 5, we have that $[\![Q]\!]^\pi \xrightarrow{[\![\alpha']\!]^\pi} [\![Q'']\!]^\pi$, i.e. $[\![Q]\!]^\pi \xrightarrow{\alpha'} [\![Q'']\!]^\pi$. We have now to prove that $[\![P'']\!]^\pi \mathcal{R} [\![Q'']\!]^\pi$. This follows easily from the fact that $P'' \leftrightarrows Q''$.

Let us now consider the case (3), namely the transition is an α-conversion transition. Since $[\![P]\!]^\pi \approx_\alpha P'$, by Lemma 10, which states the correctness of α-conversion transitions, we know that $P \equiv P''$ and $[\![P'']\!]^\pi = P'$. We now have to show that $[\![Q]\!]^\pi \approx_\alpha Q'$ for some Q' such that $P' \mathcal{R} Q'$. Since $\equiv \subset \leftrightarrows$, we have that there exists a Q'' such that $Q \equiv Q''$ and it holds that $P'' \leftrightarrows Q''$. Since $Q \equiv Q''$, by Lemma 10 we have that $[\![Q]\!]^\pi \approx_\alpha Q'$, with $[\![Q'']\!]^\pi = Q'$. We have now to prove that $[\![P'']\!]^\pi \mathcal{R} [\![Q'']\!]^\pi$. This follows easily from the fact that $P'' \leftrightarrows Q''$.

Completeness: It suffices to show that the relation

$$\mathcal{R} = \{(P, Q) \mid [\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi\}$$

is a bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $P \mathcal{R} Q$ and $P \xrightarrow{\alpha} P'$. By Theorem 5, which states the operational correctness of our formulation of π-calculus with respect to the original one, we know that $[\![P]\!]^\pi \xrightarrow{[\![\alpha]\!]^\pi} [\![P']\!]^\pi$. Since $[\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi$, we also have that $[\![Q]\!]^\pi \xrightarrow{[\![\alpha]\!]^\pi} Q''$, for some Q'' such that $[\![P']\!]^\pi \underline{\leftrightarrow}^- Q''$, and by Theorem 5 we know that Q'' is such that $Q \xrightarrow{\alpha} Q'$ with $[\![Q']\!]^\pi = Q''$. We have now to prove that $P' \mathcal{R} Q'$. This follows easily from the fact that $[\![P']\!]^\pi \underline{\leftrightarrow}^- [\![Q']\!]^\pi$.

J Open bisimilarity and Bisimilarity coincide: Proof of Theorem 7

In this section we prove that what open bisimilarity does in the ordinary early π-calculus is exactly what nominal bisimilarity does in our formulation of the early π-calculus in the nominal SOS framework.

In order to prove this, we first define an equivalence relation over terms of the ordinary π-calculus, we call it *One-Step Open Bisimilarity*. This equivalence requires the ordinary bisimilarity, as recalled in Definition 15, to match also all the "single substitutions" performed from the processes to be equated. We prove that the nominal bisimilarity in our formulation of π-calculus coincides with the One-Step Open Bisimilarity. We then prove that the One-Step Open Bisimilarity is another way to formulate the open bisimilarity of Definition 18. The statement of Theorem 7 then easily follows.

Definition 21 (One-Step Open Bisimilarity) One-Step open bisimilarity $\underline{\leftrightarrow}^{1sO}$ *is the largest symmetric relation* $\sim$ *between* π*-calculus processes such that whenever* $P \sim Q$*,*

1. *for all actions* α*, if* $P \xrightarrow{\alpha} P'$*, then there exists some* Q'*, such that* $Q \xrightarrow{\alpha} Q'$ *and* $P' \sim Q'$*;*
2. *for all channel names* a *and* b*,* $P[b/a] \sim Q[b/a]$*.*

Theorem 14 (One-Step Open bisimilarity and Bisimilarity coincide). *For all* $P, Q \in \Pi$*,* $P \underline{\leftrightarrow}^{1sO} Q$ *if, and only if,* $[\![P]\!]^\pi \leftrightarrows [\![Q]\!]^\pi$*.*

The proof of this theorem is divided into two cases:

1. Soundness: if $P \underline{\leftrightarrow}^{1sO} Q$ then $[\![P]\!]^\pi \leftrightarrows [\![Q]\!]^\pi$.
2. Completeness: if $[\![P]\!]^\pi \leftrightarrows [\![Q]\!]^\pi$ then $P \underline{\leftrightarrow}^{1sO} Q$.

Soundness: It suffices to show that the relation

$$\mathcal{R} = \{([\![P]\!]^\pi, [\![Q]\!]^\pi) \mid P \underline{\leftrightarrow}^{1sO} Q\}$$

is a nominal bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $[\![P]\!]^\pi \mathcal{R} [\![Q]\!]^\pi$ and $[\![P]\!]^\pi \xrightarrow{\alpha} P'$. In our formulation, the label α can perform be (1) an ordinary action, (2) a substitution transition, or (3) an α-conversion transition. For cases (1) and (3) the exact reasoning employed for them in the proof of Theorem 6 applies. It thus suffices considering the case (2), which is about substitution transitions. In order to prove this, we rely on Lemma 9 which states the correctness of substitution transitions, and prove that for all names a and b, $[\![P]\!]^\pi \xrightarrow{a \mapsto b} [\![P[b/a]]\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{a \mapsto b} [\![Q[b/a]]\!]^\pi$, with $[\![P[b/a]]\!]^\pi \leftrightarrows [\![Q[b/a]]\!]^\pi$. The reader can easily notice that this follows immediately. Given atoms a and b, by Clause 2 of Definition 21 we know that $P[b/a] \underline{\leftrightarrow}^{1sO} Q[b/a]$, thus $[\![P[b/a]]\!]^\pi \mathcal{R} [\![Q[b/a]]\!]^\pi$.

Completeness: It suffices to show that the relation

$$\mathcal{R} = \{(P, Q) \mid [\![P]\!]^\pi \underline{\leftrightarrow}^- [\![Q]\!]^\pi\}$$

is a one-step open bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $P\mathcal{R}Q$.

Clause 1 of Definition 21 requires that P and Q would match transitions. For this case the exact reasoning employed in the proof of Theorem 6 applies.

Clause 2 of Definition 21 concerns substitutions. We have to prove that for all channel names a and b $P[b/a]\mathcal{R}Q[b/a]$. Let us pick two names a and b, by Lemma 9 which states the correctness of substitution transitions, we know that $[\![P]\!]^\pi \xrightarrow{a\mapsto b} [\![P[b/a]]\!]^\pi$ and $[\![Q]\!]^\pi \xrightarrow{a\mapsto b} [\![Q[b/a]]\!]^\pi$. Since $[\![P]\!]^\pi \leftrightarrow [\![Q]\!]^\pi$ we moreover know that $[\![P[b/a]]\!]^\pi \leftrightarrow [\![Q[b/a]]\!]^\pi$. Thus, we can conclude that $P[b/a]\mathcal{R}Q[b/a]$.

Theorem 15 (Open and One-Step Open bisimilarity coincide).
For all $P, Q \in \Pi$, $P \underline{\leftrightarrow}^o Q$ if, and only if, $P \underline{\leftrightarrow}^{1sO} Q$.

Before embarking ourselves in the proof of Theorem 15, some preliminary considerations are in order. The completeness part of the proof of Theorem 15 relies on the fact that we can faithfully simulate the substitutions involved in the open bisimilarity, see Definition 17, by means of a sequence of substitutions that replace only one name with another. However, it is to be noticed that the substitutions of Definition 17 defines a mapping that replaces the names of a process simultaneously. For this reason, these substitutions will be called from now onwards *simultaneous substitutions*. The substitutions that replace only one atom with another within a process will be referred to as *one-step substitutions*.

The reader must see that an encoding from simultaneous to one-step substitutions cannot be provided naively. Consider for instance the term $a.b.c$[11] and a substitution σ which maps the name a to b[12], b to c, c to a and is the identity over all other atoms. The scenario we have is the following

$$(a.b.c)\sigma = b.c.a \qquad a.b.c[b/a][c/b][a/c] = a.a.a$$

The substitution σ indeed replaces names in the processes simultaneously. This fact cannot be simulated naively by one-step substitutions because of clashes of names in the terms. The unfolded procedure of the example above is $a.b.c[b/a] = b.b.c$, $b.b.c[c/b] = c.c.c$, $c.c.c[a/c] = a.a.a$. In the encoding of simultaneous substitutions that we present, we are able to simulate substitutions relying, not surprisingly, on freshness of atoms, which once again plays a crucial role.

In order to ease the proof and its presentation, we adopt a convenient representation of simultaneous substitutions. By Definition 17, we have that this type of substitutions act only on a finite set of names, we can thus represent mappings

[11] In order to make the example clear we consider the term $a.b.c$ rather than a π-calculus process.

[12] When we say that σ maps a to b we mean $\sigma(a) = b$, i.e., the name a will be replaced by b.

as finite lists of substitutions $\{a/b\}$, with a and b names. For instance, the substitution mapping a to b, c to d, e to f and that is the identity over all other names, is represented as $\{a/b\} \cdot \{c/d\} \cdot \{e/f\} \cdot \epsilon$. Such a representation gives a clearer and more immediate presentation. The symbol ϵ denotes the empty substitution for both the types of substitution. The operation $\cdot$ denotes the composition for simultaneous substitutions. For one-step substitution we write instead $t[a/b][c/d]$ as before. In what follows, we sometimes omit writing the substitution ϵ at the end of a composition of simultaneous substitutions.

Despite the representation, the reader must keep in mind that a substitution acts as a mapping and it performs simultaneous substitutions, as in the example above.

In what follows, we say that a name a is *fresh* in a simultaneous substitution σ if a does not appear in σ.

Since the correctness of the following encoding is built upon freshness of names of the term the substitution is applied to, the encoding is parametrized by a process P. Moreover, the encoding is parametrized also by an enumeration Φ of names of π-calculus. Note that this is possible because the set of names is countably infinite. The role of Φ will be make clear after the presentation of the encoding.

Definition 22 (Encoding of simultaneous substitutions)

$$\begin{aligned} [\![\epsilon]\!]_{(P,\Phi)} &= \epsilon \\ [\![\{b/a\} \cdot \sigma]\!]_{(P,\Phi)} &= [a_f/a][\![\sigma]\!]_{P[a_f/a]}[b/a_f] \end{aligned}$$

Where the name a_f is such that $\Phi(n) = a_f$ for some n that is the least natural number m such that $\Phi(m) = a_f$ and a_f is fresh in P and in $\{b/a\} \cdot \sigma$.

In the encoding above $P[a_f/a]$ is performed by the one-step substitution.

Remark on the enumeration. The encoding could simply pick a name a_f that is fresh in P and in $\{b/a\} \cdot \sigma$. However, this choice makes the encoding able to produce different outputs depending on the choice of fresh names made at any step, i.e., the encoding would not be a function. It is however convenient to avoid the technicalities that arise with dealing with multiple encodings. To this aim, we fix an enumeration of names and pick always the least suitable fresh name (with the property stated in Definition 22). It is not hard to see that such a suitable name always exists. The encoding $[\![\cdot]\!]_{(P,\Phi)}$ is thus a function.

From now onwards, the choice of the enumeration Φ will always be irrelevant, and we will just write $[\![\sigma]\!]_P$ mentioning only the parameter P.

For the sake of example, the encoding for the simultaneous substituton of the previous example is $[\![\{b/a\}\cdot\{c/b\}\cdot\{a/c\}]\!]_{(a.b.c)} = [a_f/a][b_f/b][c_f/c][a/c_f][c/b_f][b/a_f]$. The reader may want to see these six one-step substitutions applied to the term $a.b.c$:

(1) $a.b.c[a_f/a] = a_f.b.c$ (2) $a_f.b.c[b_f/b] = a_f.b_f.c$ (3) $a_f.b_f.c[c_f/c] = a_f.b_f.c_f$
(4) $a_f.b_f.c_f[a/c_f] = a_f.b_f.a$ (5) $a_f.b_f.a[c/b_f] = a_f.c.a$ (6) $a_f.c.a[b/a_f] = b.c.a$.

The series of one-step substitutions ends up in the term $b.c.a$, as we expected. This is not by chance, the following theorem proves that, thanks to the encoding $[\![\cdot]\!]$, one-step substitutions are able to simulate simultaneous substitutions.

Multiple representations for simultaneous substitutions Before stating the theorem, the reader must know that the mappings of Definition 17 allow for multiple representations of simultaneous substitutions. For instance, the two substitutions $\{a/b\} \cdot \{c/d\} \cdot \{e/f\}$ and $\{a/b\} \cdot \{e/f\} \cdot \{c/d\}$ represent the same mapping. We equate all of these representations, and when refering to a simultaneous substitution σ, we actually refer to a representative representation of the class of all the simultaneous substitutions which differs only by permutation of their single substitutions. It is not hard to see that all of these equated representations lead to the same term when applied to a process P.

Theorem 16 (Correctness of the encoding of simultaneous substitutions).
For all processes P in Π, for all simultaneous substitutions σ, $P\sigma = P[\![\sigma]\!]_P$.

The proof of Theorem 16 can be found in Section K. Relying on Theorem 16, we proceed to prove Theorem 15. Before embarking on the proof, we state a useful lemma.

The following lemma, whose proof is straightforward, ensures that simultaneous substitutions and one-step substitutions coincide when substituting only one name.

Lemma 14. *For all $P \in \Pi$, for all names a and b it holds that $P\{a/b\} = P[a/b]$.*

The proof is divided into two cases, given P and $Q \in \Pi$:

1. Soundness: if $P \underline{\leftrightarrow}^o Q$ then $P \underline{\leftrightarrow}^{1sO} Q$.
2. Completeness: if $P \underline{\leftrightarrow}^{1sO} Q$ then $P \underline{\leftrightarrow}^o Q$.

Soundness: It suffices to show that the relation

$$\mathcal{R} = \{(P, Q) \mid P \underline{\leftrightarrow}^o Q\}$$

is a one-step open bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $P \underline{\leftrightarrow}^o Q$. Consider first Clause 1 of Definition 21 and assume $P \xrightarrow{\alpha} P'$. Since $\underline{\leftrightarrow}^o$ ranges over all the simulation substitutions, it ranges also over the simultaneous substitution ι which is the identity over all the names, i.e., such that $P\iota = P$ for all $P \in \Pi$. We have thus that $P\iota \xrightarrow{\alpha} P'$. Since $P \underline{\leftrightarrow}^o Q$, we have that there exists a process Q' such that $Q\iota \xrightarrow{\alpha} Q'$, which simply means $Q \xrightarrow{\alpha} Q'$, and $P' \underline{\leftrightarrow}^o Q'$. We now have to prove that $P' \mathcal{R} Q'$. This follows from the fact that $P' \underline{\leftrightarrow}^o Q'$.

Consider now Clause 2 of Definition 21, which is about one-step substitutions. It is easy to see that since $\underline{\leftrightarrow}^o$ ranges over all the simultaneous substitutions, it also ranges over all the simultaneous substitutions that replace only one name with another. By Lemma 14, these substitutions coincide to their corresponding one-step substitutions.

Completeness: It suffices to show that the relation
assume $P\sigma \xrightarrow{\alpha} P'$. L

$$\mathcal{R} = \{(P, Q) \mid P \underline{\leftrightarrow}^{1sO} Q\}$$

is an open bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric. Assume now that $P \underline{\leftrightarrow}^{1sO} Q$. Let us pick a simultaneous substitutions σ and let us consider its encoding as a series of -one-step substitutions $\sigma^* = [\![\sigma]\!]_{(P \,||\, Q)}$ (We use $P \,||\, Q$ as parameter in order to make sure that the encoding will pick names that are fresh in both P and Q). Note that if a name is fresh in both P and Q, it is fresh in P and Theorem 16 applies. The same holds for Q. Let σ^* be $[a_1/a_2][a_3/a_4] \cdot [a_{n-1}, /a_n] \cdot \epsilon$. The first one-step substitution $[a_1/a_2]$ is considered by $\underline{\leftrightarrow}^{1sO}$, and we have that $P[a_1/a_2] \underline{\leftrightarrow}^{1sO} Q[a_1/a_2]$. Now, since $P[a_1/a_2] \underline{\leftrightarrow}^{1sO} Q[a_1/a_2]$, we have that, again by Clause 2 of Definition 21, the substitution $[a_3, a_4]$ is considered by $\underline{\leftrightarrow}^{1sO}$ from the terms $P[a_1/a_2]$ and $Q[a_1/a_2]$, and $P[a_1/a_2][a_3/a_4] \underline{\leftrightarrow}^{1sO} Q[a_1/a_2][a_3, a_4]$. By iterating the application of Clause 2 of Definition 21, we can apply to P and Q the complete sequence of one-step substitutions σ^*, ending up with $P\sigma^* \underline{\leftrightarrow}^{1sO} Q\sigma^*$. Now, thanks to the Clause 1 of Definition 21, since $P\sigma^* \underline{\leftrightarrow}^{1sO} Q\sigma^*$, if $P\sigma^* \xrightarrow{\alpha} P'$ then $Q\sigma^* \xrightarrow{\alpha} Q'$ and $P' \underline{\leftrightarrow}^{1sO} Q'$. By Theorem 16, $P\sigma^* = P\sigma$, so also $P\sigma$ is such that $P\sigma \xrightarrow{\alpha} P'$. For the same reasons, we have that also $Q\sigma$ is such that $Q\sigma \xrightarrow{\alpha} Q'$. Now, since $P' \underline{\leftrightarrow}^{1sO} Q'$ we have that $P'\mathcal{R}Q'$.

This concludes the proof of Theorem 15. The statement of Theorem 7 easily follows from Theorem 14 and Theorem 15.

K Simulation of Substitutions by One-Step Substitutions: Proof of Theorem 16

The proof of Theorem 16 relies on two lemmas, whose proofs are omitted.

Lemma 15. *For all $P \in \Pi$, for each simultaneous substitution σ and names a and b, if σ does not map a, b is fresh in P and b does not appear in σ, then it holds that $(P\{b/a\})\sigma = P\{b/a\} \cdot \sigma$.*

The reader must notice that differently from what happens in $(P\{b/a\})\sigma$, in the term $P\{b/a\} \cdot \sigma$ the substitution $\{b/a\}$ is performed simultaneously with the others of σ. The proof of Lemma 15 is straightforward and omitted.

The following lemma says, instead, that in order to change an atom a into b in a term while performing other substitutions, we can first change the atom a into a fresh new atom a_f and apply simultaneously the further substitutions, and after that, replace the atom a_f with b. We can do this as long as the further substitutions do not change a_f or introduce other occurrences of a_f.

Lemma 16. *For all $P \in \Pi$, for each simultaneous substitution σ and names a, b and a_f, if σ does not map a and a_f is fresh in P and a_f does not appear in σ, then it holds that $P\{b/a\} \cdot \sigma = (P(\{a_f/a\} \cdot \sigma))\{b/a_f\}$.*

Let P be a process of the π-calculus and σ be a simultaneous substitution. The proof proceeds by induction on the "length" of σ, i.e., the cardinality of the set of atoms over which σ does not act as the identity. Note that by Definition 17 this set is finite. For what follows, we invite the reader to pay attention to the fact that the two expressions $(P\{a_f/a\})\sigma$ and $P(\{a_f/a\}\cdot\sigma)$ mean different things. The former denotes the substitution σ applied to the term $P\{a_f/a\}$, while the latter denotes the substitution $\{a_f/a\}\cdot\sigma$ applied to the term P.

- Length 0, i.e., $\sigma = \epsilon$: It is easy to see that $P\epsilon = P[\![\epsilon]\!]_P = \epsilon$.
- Length $n > 0$, i.e., $\sigma = \{b/a\}\cdot\sigma'$: Then, $[\![\sigma]\!]_P = [a_f/a][\![\sigma']\!]_{P[a_f/a]}[b/a_f]$, with a_f fresh in P. Now, let us consider the term $P\{a_f/a\}$. By inductive hypothesis, which apply to σ', $(P\{a_f/a\})[\![\sigma']\!]_{P\{a_f/a\}} = (P\{a_f/a\})\sigma'$. By Lemma 14, we know that $P\{a_f/a\} = P[a_f/a]$, so we have $(P[a_f/a])[\![\sigma']\!]_{P[a_f/a]} = (P\{a_f/a\})\sigma'$. Again, by Lemma 14, we can add one single additional substitution without affecting the equation, so we have $((P[a_f/a])[\![\sigma']\!]_{P[a_f/a]})[b/a_f] = ((P\{a_f/a\})\sigma')\{b/a_f\}$. Since σ is a mapping, clearly σ' does not map a. We know also that a_f does not appear in σ' and that a_f is fresh in P. We can thus apply Lemma 15 and have that $(P\{a_f/a\})\sigma' = P\{a_f/a\}\cdot\sigma'$. The equation thus rewrites as $P[a_f/a][\![\sigma']\!]_{P[a_f/a]}[b/a_f] = (P\{a_f/a\}\cdot\sigma')\{b/a_f\}$. Again, since σ' does not map a, a_f does not appear in σ' and a_f is fresh in P, we can apply Lemma 16 and have that $(P\{a_f/a\}\cdot\sigma')\{b/a_f\} = P\{b/a\}\cdot\sigma'$. We can thus conclude that $P[a_f/a][\![\sigma']\!]_{P[a_f/a]}[b/a_f] = P\{b/a\}\cdot\sigma'$.

L Nominal bisimilarity equates too much in λ-calculus: Proof of Theorem 9

Before embarking on the proof of Theorem 9, let us first state some lemmas.

Lemma 17 (Free atoms of a term after a substitution). *Let T be an NTSS and let M be a term over the signature of T. For all atoms b, it holds that:*

- *If $fa(M) = \Phi$ and $a \notin \Phi$, then $fa(M[b/a]) = \Phi$ (if $M[b/a]$ is defined).*
- *If $fa(M) = \Phi \cup \{a\}$ and $a \notin \Phi$, then $fa(M[b/a]) = \Phi \cup \{b\}$ (if $M[b/a]$ is defined).*

Lemma 18 (Substituting a free atom with a binding-closed term). *Consider the NTSS of our lazy λ-calculus defined in Section 5.1. Let $M \in \mathbb{C}(\Sigma^\lambda)$ and let $fa(M) = \{a\}$, for some atom a. For all terms $M' \in \mathbb{C}(\Sigma^\lambda)$ and $N \in \mathbb{C}(\Sigma^\lambda)^0$, it holds that if $M \overset{a \overset{T}{\mapsto} N}{\longrightarrow} M'$ then $M' \in \mathbb{C}(\Sigma^\lambda)^0$.*

Lemmas 17 and 18 can both be proved by an induction on the structure of M. The proofs are simple and omitted. Note, however, that these proofs rely on the straightforward fact that $a \notin fa([a]M)$ for every atom a and term M.

Lemma 19 (α-conversions preserve the set of free atoms). *Let T be an NTSS whose set of rules contains the rules for α-conversion transitions as defined in Section 4.2. Let M be a term over the signature of T and let $fa(M) = \Phi$. For all terms N, it holds that if $M \approx_\alpha N$ then $fa(N) = \Phi$.*

Intuitively, it is easy to see from the rules of Section 4.2 that an α-conversion transition does not introduce or delete free atoms in a term. Lemma 19 can be proved by an induction on the length of the proofs of provable α-conversion transitions. However, such a proof makes use of a technical detail for the case of terms of form $[a].M$. We shall discuss only this case in detail. There are two possible α-conversion transitions from the term $[a].M$. Namely (1) $[a]M \approx_\alpha [b]M'$ by rule $(abs1_\alpha)$, with $b\#M$ and $M \xrightarrow{a \leftrightarrow b}_A M'$, and (2) $[a]M \approx_\alpha [a]M'$ by rule $(abs2_\alpha)$, with $M \approx_\alpha M'$.

Let us consider (1). Let $fa([a].M) = \Phi$. By definition of the set fa, $a \notin \Phi$. We also have that $fa(M) = \Phi$ or $fa(M) = \Phi \cup \{a\}$. Now, By Lemma 2, which states the correctness of substitution transitions, we know that $M' = M[b/a]$. By Lemma 17 we can conclude that $fa(M[b/a]) = \Phi$ and $b \notin \Phi$, or $fa(M[b/a]) = \Phi \cup \{b\}$. Either way, when we abstract on the atom b from $M[b/a]$, we have $fa([b].M[b/a]) = \Phi$.

Let us consider (2). Let $fa([a].M) = \Phi$, it holds again that $a \notin \Phi$ and that $fa(M) = \Phi$, or $fa(M) = \Phi \cup \{a\}$. Now, since the proof length for proving the transition $M \approx_\alpha M'$ is strictly less than the proof length for proving $[a]M \approx_\alpha [a]M'$, by the inductive hypothesis we can conclude that $fa(M') = \Phi$ or $fa(M') = \Phi \cup \{a\}$. Either way, when we abstract on the atom a from M', we have $fa([a].M') = \Phi$.

As a straightforward consequence of Lemma 19, α-conversion transitions preserve binding-closedness of terms, as stated below.

Lemma 20 (α-conversions preserve binding-closedness of terms). *Let T be an NTSS with signature Σ and let the set of rules of Σ contain the rules for α-conversion transitions as defined in Section 4.2. Let $M \in \mathbb{C}(\Sigma)^0$. For all terms N, it holds that if $M \approx_\alpha N$ then $N \in \mathbb{C}(\Sigma)^0$.*

For any two α-equivalent terms of our formulation of the lazy λ-calculus, it holds that either they both have a normal form or they both do not have a normal form, as stated below.

Lemma 21 (α-conversions preserve the normal form). *Consider the NTSS of our lazy λ-calculus defined in Section 5.1. Let M be a term over Σ^λ and let M have a normal form. For all terms N, it holds that if $M \approx_\alpha N$ then N has a normal form.*

Lemma 21 follows easily from the fact that the reduction $\rightarrow$ is up to α-equivalence and the transition $\approx_\alpha$ is 'symmetric', as shown in the proof of Theorem 3 in Section **??**. These two facts imply that if the transition $M \rightarrow M'$ is provable, then also the transition $N \rightarrow M'$ is provable, by α-converting first N back to M.

Lemma 22 (Substitutions are always possible on λ-terms). *Consider the NTSS of our lazy λ-calculus defined in Section 5.1. Let $M \in \mathbb{C}(\Sigma^{\lambda})$. For all atoms a and terms N, it holds that $M \stackrel{a^{\overset{T}{\mapsto}N}}{\longrightarrow} M'$ for some term M'.*

Lemma 22 can be proved by an induction on the length of the proofs of provable substitution transitions. The only delicate point is when performing a substitution $\stackrel{a^{\overset{T}{\mapsto}N}}{\longrightarrow}$ in a term M and M contains an abstraction $[b].M_1$ with b not fresh in N. However, as argued in Section 5.3, since we set the term-for-atom substitutions of our lazy λ-calculus to be up to α-equivalence, a substitution transition is possible also in this case.

By way of example, consider the provable transition

$$(\lambda([a].\lambda([b].(b\ a)))\ \stackrel{c^{\overset{T}{\mapsto}(b\ b)}}{\longrightarrow}\ (\lambda([a].\lambda([d].(d\ a))),$$

where d is fresh in $(b\ a)$. In this case an α-conversion takes place before the application of rule $(abs1_{\mathrm{Ts}})$.

This point is not shown formally here. However, it follows exactly the same reasoning employed in case 3 of the enumeration that contains the proofs of a few cases for Lemma 23 below.

Lemma 23 (Substitutions are 'ineffective' on binding-closed terms). *Consider the NTSS of our lazy λ-calculus defined in Section 5.1. Let $M \in \mathbb{C}(\Sigma^{\lambda})^0$ and let b be an atom fresh in M. For all terms N and M', it holds that if $M \stackrel{b^{\overset{T}{\mapsto}N}}{\longrightarrow} M'$ then $M \approx_{\alpha} M'$.*

Intuitively, it is easy to see that the substitution transitions defined by the rules of Section 4.1 can only substitute free atoms in a term. Since a binding-closed term contains no free atom, a substitution ends up in the same term up to α-equivalence.

Lemma 23 can be proved by an induction on the length of the proofs of provable substitution transitions. However, the cases involving an abstraction $[a].M$ are quite delicate and we wish to discuss them in detail.

1. Let us consider the term $[a]M$ and the substitution transition $\stackrel{b^{\overset{T}{\mapsto}N}}{\longrightarrow}$ where $b \neq a$ and a is fresh in N. By the proviso of the lemma, we have also that b is fresh in $[a]M$. In this case we can apply only the rule $(abs1_{\mathrm{Ts}})$, as $(abs2_{\mathrm{Ts}})$ requires that $a = b$. The rule $(abs1_{\mathrm{Ts}})$ is instanciated as follows.

$$\frac{M \stackrel{b^{\overset{T}{\mapsto}N}}{\longrightarrow} M' \quad a\#N \quad a \neq b}{[a]M \stackrel{b^{\overset{T}{\mapsto}N}}{\longrightarrow} [a]M'}$$

   Notice that the premises $a\#N$ and $a \neq b$ are satisfied. Moreover, since b is fresh in $[a]M$, then $b \notin fa(M)$. Indeed, the term M may have only the atom

a as free, if any. Since b is fresh in M, and since the proof length for proving the transition $M \stackrel{b\overset{T}{\mapsto}N}{\longrightarrow} M'$ is strictly less than the proof length for proving $[a]M \stackrel{b\overset{T}{\mapsto}N}{\longrightarrow} [a]M'$, the inductive hypothesis the inductive hypothesis applies and $M \approx_\alpha M'$. Therefore, the transition proved above is $[a]M \stackrel{b\overset{T}{\mapsto}N}{\longrightarrow} [a]M'$, with $[a]M \approx_\alpha [a]M'$ (by rule $(abs2_\alpha)$).

2. Let us consider the term $[a].M$ and the substitution transition $\stackrel{a\overset{T}{\mapsto}N}{\longrightarrow}$ (it is irrelevant whether a is fresh in N or not). In this case we cannot apply the rule $(abs1_{\mathrm{Ts}})$ above, as the premise $a \neq b$ is not satisfied. However, we can apply the rule $(abs2_{\mathrm{Ts}})$ and prove the transition $[a]M \stackrel{a\overset{T}{\mapsto}N}{\longrightarrow} [a]M$, for which it clearly holds that $[a]M \approx_\alpha [a]M$.
3. The case discussed in this item is related to the case pointed out for Lemma 22 above. Let us consider the term $[a].M$ and the substitution transition $\stackrel{b\overset{T}{\mapsto}N}{\longrightarrow}$ where $b \neq a$ and a is *not* fresh in N. By the proviso of the lemma, we have also that b is fresh in $[a]M$. In this case we cannot apply the rule $(abs1_{\mathrm{Ts}})$, as it requires that $a\#N$. Also, we cannot apply the rule $(abs2_{\mathrm{Ts}})$, as it requires that $a = b$. However, since the term-for-atom substitution transitions are set to be up to α-equivalence, we can pick an atom c that is fresh in N and instanciate the rule $(b \overset{T}{\mapsto} N \cdot \mathrm{upTo}\alpha)$ as follows.

$$\frac{[a]M \approx_\alpha [c]M' \quad [c]M' \stackrel{b\overset{T}{\mapsto}N}{\longrightarrow} [d]M''}{[a]M \stackrel{b\overset{T}{\mapsto}N}{\longrightarrow} [d]M''}.$$

Now, by Lemma 19, since $[a]M \approx_\alpha [c]M'$ we have that $fa([a]M) = fa([c]M')$. This means that since b is fresh in $[a]M$ then b is fresh also in $[c]M'$. Given this fact, and also since the proof length for proving the transition $[c]M' \stackrel{b\overset{T}{\mapsto}N}{\longrightarrow} [d]M''$ is strictly less than the proof length for proving $[a]M \stackrel{b\overset{T}{\mapsto}N}{\longrightarrow} [d]M''$, the inductive hypothesis applies and we have that $[c]M' \approx_\alpha [d]M''$. Since $[a]M \approx_\alpha [c]M'$ is provable and also $[c]M' \approx_\alpha [d]M''$ is provable, we can use the rule $(\alpha \cdot \mathrm{upTo}\alpha)$, (basically closing by transitivity), and prove the transition $[a]M \approx_\alpha [d]M''$ as required.

The reader may benefit also from an informal intuition on the dynamics at play in the cases above. The enumeration below contains useful examples and will serve this purpose. Note that, each number of the enumeration for the examples below corresponds to the number of the enumeration of the cases proved above. In what follows, atoms with distinct names are considered different atoms.

1. $(\lambda([a].\lambda([b].(b\ a)))\ \stackrel{c\overset{T}{\mapsto}(d\ d)}{\longrightarrow}\ (\lambda([a].\lambda([b].(b\ a)))$, employing the rule $(\lambda\mathrm{Ts})$ with $(abs1_{\mathrm{Ts}})$.

2. $(\lambda([a].\lambda([b].(b\ a)))\ \stackrel{a\stackrel{T}{\mapsto}(d\ d)}{\longrightarrow}\ (\lambda([a].\lambda([b].(b\ a)))$, employing the rule ($\lambda$Ts) with ($abs2_{\mathrm{Ts}}$). Employing the same rules we can prove also $(\lambda([a].\lambda([b].(b\ a)))\ \stackrel{a\stackrel{T}{\mapsto}(b\ b)}{\longrightarrow}\ (\lambda([a].\lambda([b].(b\ a)))$.
3. $(\lambda([a].\lambda([b].(b\ a)))\ \stackrel{c\stackrel{T}{\mapsto}(b\ b)}{\longrightarrow}\ (\lambda([a].\lambda([d].(d\ a)))$, where d is fresh in $(b\ a)$. This case is related to Lemma 22.

Lemma 24 (Reductions preserve binding-closedness and the normal form). *Consider the NTSS of our lazy λ-calculus defined in Section 5.1. Let $M \in \mathbb{C}(\Sigma^\lambda)^0$ have a normal form. For all terms M', it holds that if $M \to M'$ then $M' \in \mathbb{C}(\Sigma^\lambda)^0$ and M' has a normal form.*

Lemma 24 can be proved by an induction on the length of the proofs of provable transitions $\to$. We single out only the case when the rule (*app*) is applied. Let us assume that $(M\ N) \to M'''$ is provable by the rule (*app*) instanciated as follows.

$$\frac{M \to \lambda([a]M') \quad M' \stackrel{a\stackrel{T}{\mapsto}N}{\longrightarrow} M'' \quad M'' \to M'''}{(M\ N) \to M'''}\ (\mathrm{app})$$

Now, since $(M\ N) \in \mathbb{C}(\Sigma^\lambda)^0$, we have that $M \in \mathbb{C}(\Sigma^\lambda)^0$ and $N \in \mathbb{C}(\Sigma^\lambda)^0$. As the transition $M \to \lambda([a]M')$ is provable, M has a normal form. Since also $M \in \mathbb{C}(\Sigma^\lambda)^0$ and the fact that the proof length for proving the transition $M \to \lambda([a]M')$ is strictly less than the proof length for proving $(M\ N) \to M'''$, the inductive hypothesis applies and we have that $\lambda([a]M') \in \mathbb{C}(\Sigma^\lambda)^0$. This means that $fa(M') = \emptyset$ or $fa(M') = \{a\}$. If $fa(M') = \emptyset$, then $M' \in \mathbb{C}(\Sigma^\lambda)^0$ and by Lemmas 23 and 19 we can conclude that after the transition $M' \stackrel{a\stackrel{T}{\mapsto}N}{\longrightarrow} M''$ we have that $M'' \in \mathbb{C}(\Sigma^\lambda)^0$ (by applying also Lemma 20 to M''). If $fa(M') = \{a\}$, since $N \in \mathbb{C}(\Sigma^\lambda)^0$, we can apply Lemma 18 and have that $M'' \in \mathbb{C}(\Sigma^\lambda)^0$. Either way, we have that $M'' \in \mathbb{C}(\Sigma^\lambda)^0$. Now, since the transition $M'' \to M'''$ is provable, M'' has a normal form. Since also $M'' \in \mathbb{C}(\Sigma^\lambda)^0$ and the fact that the proof length for proving the transition $M'' \to M'''$ is strictly less than the proof length for proving $(M\ N) \to M'''$, the inductive hypothesis applies and we have that $M''' \in \mathbb{C}(\Sigma^\lambda)^0$ and has a normal form.

The rest of the proof proceeds following standard lines. Note, however, that the proof for transitions $M \to M'$ up to α-conversion, i.e. an α-conversion takes place first, makes use of Lemmas 20 and 21, which state that α-conversions preserve binding-closedness and 'having a normal form', respectively.

We are now ready to prove Theorem 9. To this end, it is sufficient to show that the relation

$$\mathcal{R} = \{(M, N) \mid M, N \in \mathbb{C}(\Sigma^\lambda)^0 \textit{ and } M, N \textit{ have normal } form\}$$

is a nominal bisimulation.

Notice first of all that $\mathcal{R}$ is symmetric. Assume now that $M\ \mathcal{R}\ N$. There are three types of transitions that we need to consider within the bisimulation

game of Definition 16. Namely, (1) substitutions transitions, (2) α-conversion transitions and (3) transitions performed by the reduction step $\rightarrow$.

Let us consider first the case (1). Let us pick an atom a and a term P and assume that $M \stackrel{a \stackrel{T}{\mapsto} P}{\longrightarrow} M'$, for some term M'. Since $M \in \mathbb{C}(\Sigma^\lambda)^0$, by Lemma 23 we know that $M \approx_\alpha M'$. Since $M \in \mathbb{C}(\Sigma^\lambda)^0$ and M has a normal form, by Lemmas 20 and 21 we can conclude that $M' \in \mathbb{C}(\Sigma^\lambda)^0$ and also that M' has a normal form. Now, since $N \in \mathbb{C}(\Sigma^\lambda)^0$, by Lemma 22 substitutions are always possible and we have that $N \stackrel{a \stackrel{T}{\mapsto} P}{\longrightarrow} N'$ for some term N'. By Lemma 23, we also know that $N \approx_\alpha N'$. Again, since $N \in \mathbb{C}(\Sigma^\lambda)^0$ and N has a normal form, by Lemmas 20 and 21 we can conclude that $N' \in \mathbb{C}(\Sigma^\lambda)^0$ and also that N' has a normal form. We therefore have that $M' \mathcal{R} N'$.

Let us consider now the case (2) and assume that $M \approx_\alpha M'$, for some term M'. Using Lemmas 20 and 21, we know that $M' \in \mathbb{C}(\Sigma^\lambda)^0$ and that M' has a normal form. Now, we can use the rule (id_α) in order to prove the transition $N \approx_\alpha N$. Since $N \in \mathbb{C}(\Sigma^\lambda)^0$ and N has a normal form, we have therefore that $M' \mathcal{R} N$.

Let us consider now the case (3) and assume that $M \rightarrow M'$, for some term M'. By Lemma 24 we know that $M' \in \mathbb{C}(\Sigma^\lambda)^0$ and that M' has a normal form. Now, since N has a normal form, we have that we can prove a transition $N \rightarrow N'$, for some term N'. Moreover, by Lemma 24 we have that $N' \in \mathbb{C}(\Sigma^\lambda)^0$ and also that N' has a normal form. We have therefore that $M' \mathcal{R} N'$. This concludes the proof.

M Applicative and nominal bisimilarity coincide in the alternative formulation of the λ-calculus.

In order to establish a correspondence between applicative bisimilarity in the original lazy λ-calculus and nominal bisimilarity in the nominal formulation of it presented in Section 7.1, we need to establish the correctness of substitutions and α-conversion transitions as well as of transitions $\rightarrow$.

Notice that, since the formulation of the lazy λ-calculus employs the signature Σ^λ, the correctness of substitutions and α-conversion transitions has already been established by Lemma 5 and Lemma 4, respectively.

The following lemma states that a transition $\lambda([a].M) \stackrel{P}{\rightarrow} M'$ corresponds to feeding the abstraction $\lambda([a].M)$ with the term P.

Lemma 25. *For all λ-terms M, binding-closed λ-terms P and atoms a, it holds that*

- $[\![\lambda a.M]\!]^\lambda \stackrel{[\![P]\!]^\lambda}{\rightarrow} [\![M[P/a]]\!]^\lambda$.
- *if* $[\![\lambda a.M]\!]^\lambda \stackrel{[\![P]\!]^\lambda}{\rightarrow} N$ *then* $N = [\![M[P/a]]\!]^\lambda$.

Lemma 25 says first that for any binding-closed λ-terms P, a transition $\stackrel{[\![P]\!]^\lambda}{\rightarrow}$ always exists from abstractions. It also says that this transition is deterministic,

a result that comes directly from the determinism of substitution transitions stated in Lemma 2. These are considerations that will be employed in the proof of the coincidence between applicative and nominal bisimilarity. Another lemma that is employed in the proof of the correctness of transitions $\rightarrow$ states that the target of this sort of transitions is always an abstraction.

Lemma 26 (Transitions $\rightarrow$ always produce an abstraction). *For all terms $M \in \mathbb{C}(\Sigma^{\lambda})^0$, it holds that if $M \rightarrow N$ then $N = \lambda([a].M')$, for some atom a and term $M' \in \mathbb{C}(\Sigma^{\lambda})$.*

The operational correspondence that we aim to prove relies also on the following straightforward observations: 1. The encoding $[\![\cdot]\!]^{\lambda}$ is surjective, i.e. every term in our formulation of the lazy λ-calculus is the the encoding of a λ-term. 2. Closed λ-terms encode into binding-closed terms and viceversa. For ease of reference, we state these lemmas below.

Lemma 27 (Surjectivity of $[\![\cdot]\!]^{\lambda}$). *For all terms $M' \in \mathbb{C}(\Sigma^{\lambda})$ of sort L, it holds that $[\![M]\!]^{\lambda} = M'$, for some $M \in \Lambda$.*

Lemma 28 (Absence of free atoms/variables is preserved by $[\![\cdot]\!]^{\lambda}$).

- *For all $M \in \Lambda^0$ it holds that $[\![M]\!]^{\lambda} \in \mathbb{C}(\Sigma^{\lambda})^0$.*
- *For all $M' \in \mathbb{C}(\Sigma^{\lambda})^0$ it holds that $[\![M]\!]^{\lambda} = M'$ for some $M \in \Lambda^0$.*

We now state the operational correspondence between our new formulation of the lazy λ-calculus and its original formulation.

Theorem 17 (Operational correspondence for the new formulation).

- *$\Rightarrow$: For all $M \in \Lambda^0$ and $N \in \Lambda$, it holds that if $M \rightarrow N$ then $[\![M]\!]^{\lambda} \rightarrow [\![N]\!]^{\lambda}$.*
- *$\Leftarrow$: For all $M \in \Lambda^0$ and $N \in \mathbb{C}(\Sigma^{\lambda})$ it holds that if $[\![M]\!]^{\lambda} \rightarrow N$ then $M \rightarrow N'$ and $[\![N']\!]^{\lambda} = N$, for some term $N' \in \Lambda^0$.*

The reader should notice that the operational correspondence just stated is weaker than the one established in Theorem 4 in the context of our first formulation of the lazy λ-calculus. Indeed, the correspondence stated in Theorem 17 only applies to closed λ-terms. The reason for such a weaker theorem is that the new formulation perform the parameter passing only for binding-closed terms. The reason of this choice is motivated in Section 5.1. The proof of the coincidence between applicative and nominal bisimilarity makes use of this fact and we shall state it explicitly below. It is also to notice that the case $\Rightarrow$ implies that

Lemma 29 (The parameter-passing involves only binding-closed terms). *For all terms M, N and $P \in \mathbb{C}(\Sigma^{\lambda})$, it holds that if $M \xrightarrow{P} N$ then $P \in \mathbb{C}(\Sigma^{\lambda})^0$.*

The proofs of Lemma 25-29 are straightforward and therefore omitted.

The proof of Theorem 17 follows the same lines of the proof of Theorem 4 and we omit the details. The proof proceeds by induction on the length of a proof of the relevant transition. The only non-trivial case in the proof of the "$\Leftarrow$" statement in the theorem is for rule (app1AP) and in that case the proof uses first Lemma 26 to relate the premise $x_0 \to y_0$ of rule (app1AP) to the premise $x_0 \to \lambda a.y_0$ of rule (appO). Then the proof proceeds from this point by using Lemma 25 for matching the premises $y_0 \xrightarrow{x_1} y_1$ and $y_1 \to y_2$ of rule (app1AP) against the premise $y_0[x_1/a] \to y_1$ of rule (appO). In particular, y_1 of rule (app1AP) corresponds to $y_0[x_1/a]$ in (appO) and y_2 of rule (app1AP) corresponds to y_1 of rule (appO). The proof then concludes with standard reasoning.

The proof of the coincidence between applicative and nominal bisimilarity also relies, among other results, on Lemma 23, which says that substitutions are 'ineffective' on binding-closed terms, and on two well-known facts from the theory of the lazy λ-calculus that say that applicative bisimilarity and term closedness are preserved under reduction. We state these two lemmas below for the sake of completeness.

Lemma 30 (Applicative bisimilarity is preserved under reduction). *For all $M, N \in \Lambda^0$, it holds that if $M \to N$ then $M \simeq N$.*

Lemma 31 (Closedness of λ-terms is preserved under reduction). *For all $M \in \Lambda^0$ and $N \in \Lambda$, it holds that if $M \to N$ then $N \in \Lambda^0$.*

We now have everything we need in order to proceed with the proof. The proof is divided into two cases, given M and $N \in \Lambda^0$:

1. Soundness: if $M \simeq N$ implies $[\![M]\!]^\lambda \leftrightarrows [\![N]\!]^\lambda$.
2. Completeness: if $[\![M]\!]^\lambda \leftrightarrows [\![N]\!]^\lambda$ then $M \simeq N$.

Soundness: It suffices to show that the relation

$$\mathcal{R} = \{([\![M]\!]^\lambda, [\![N]\!]^\lambda) \mid M \simeq N\}$$

is a nominal bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric since so is $\simeq$. Assume now that $M \simeq N$. We now have to ensure that if $[\![M]\!]^\lambda \xrightarrow{\alpha} M'$ for some M' and label α then also a transition $[\![N]\!]^\lambda \xrightarrow{\alpha} N'$ is provable for some N' such that $M' \mathcal{R} N'$.

There are a few types of transitions for which we should ensure that this matching holds, 1) substitutions transitions, 2) α-conversion transitions, 3) reduction $\to$ and 4) the new transitions of the form $\xrightarrow{P}$. Case 1) can be proved trivially thanks to the fact that substitutions are ineffective in our context, i.e. thanks to Lemma 23. Case 2) can be proved trivially thanks to the correctness of α-conversion transitions and the fact that α-equivalence is included in the nominal bisimilarity. Case 3) is also trivial thanks to the operational correspondence stated in Theorem 17 in the first place and then to fact that applicative

bisimilarity is preserved under reduction, i.e. thanks to Lemma 30. The only relevant case is for transitions of the form $\xrightarrow{P}$, which we shall consider in detail in what follows. Let us pick $P \in \Lambda^0$ and assume $[\![M]\!]^\lambda \xrightarrow{[\![P]\!]^\lambda} M''$, for some M''. Now we have to split the four cases, whether 1) M and N are both abstractions, 2) M is an abstraction and N is an application, 3) M is an application and N is an abstraction and 4) M and N are both applications.

Let us start by considering cases 1) and 2) together. Let M be an abstraction $\lambda a.M'$. Now, since rule (abs2AP) is the only rule for proving transitions of the form $\xrightarrow{P}$ from abstractions, we have that the transition $[\![\lambda a.M']\!]^\lambda \xrightarrow{[\![P]\!]^\lambda} M''$ is provable by rule (abs2AP), for some M''. We thus have that rule (abs2AP) is instantiated in the following way.

$$\frac{[\![M']\!]^\lambda \xrightarrow{a \overset{T}{\mapsto} [\![P]\!]^\lambda} M''}{[\![\lambda a.M']\!]^\lambda \xrightarrow{[\![P]\!]^\lambda} M''}$$

By Lemma 5, which states the correctness of substitutions transitions, we now know that $M'' = [\![M'[P/a]]\!]^\lambda$.

Now, let us consider case 1) and let N be an abstraction $\lambda b.N'$. We can instantiate the same rule (abs2AP) in the following way.

$$\frac{[\![N']\!]^\lambda \xrightarrow{b \overset{T}{\mapsto} [\![P]\!]^\lambda} N''}{[\![\lambda b.N']\!]^\lambda \xrightarrow{[\![P]\!]^\lambda} N''}$$

Again, by Lemma 5, we know that $N'' = [\![N'[P/b]]\!]^\lambda$. Since $\lambda a.M' \simeq \lambda b.N'$, we have that $M'[P/a] \simeq N'[P/b]$ by definition of applicative bisimilarity, see Definition 19. We thus have that $[\![\lambda a.M']\!]^\lambda \xrightarrow{[\![P]\!]^\lambda} [\![M'[P/a]]\!]^\lambda$ and $[\![\lambda b.N']\!]^\lambda \xrightarrow{[\![P]\!]^\lambda} [\![N'[P/b]]\!]^\lambda$ and, by definition of $\mathcal{R}$, it holds that $[\![M'[P/a]]\!]^\lambda \mathcal{R} [\![N'[P/b]]\!]^\lambda$.

Consider now case 2) and let N be an application $(N_1\ N_2)$. We can instantiate rule (app2AP) as follows.

$$\frac{[\![(N_1\ N_2)]\!]^\lambda \to N' \qquad N' \xrightarrow{[\![P]\!]^\lambda} N''}{[\![(N_1\ N_2)]\!]^\lambda \xrightarrow{[\![P]\!]^\lambda} N''}$$

We know that the transition $[\![(N_1\ N_2)]\!]^\lambda \to N'$ is provable because $\lambda a.M'$ has a normal form and the two terms $\lambda a.M'$ and $(N_1\ N_2)$ are applicative bisimilar. This means that $(N_1\ N_2)$ cannot diverge and it, too, has a normal form M''. By Theorem 17 which states the operational correctness of transitions $\to$, we therefore have $[\![(N_1\ N_2)]\!]^\lambda \to N'$. By Lemma 26, which states that a reduction always produce an abstraction, we know that M'' is an abstraction. By Lemma 27 (surjectivity of $[\![\cdot]\!]^\lambda$) we know that N' is an abstraction and by a simple inspection over the encoding we can prove that N' is the encoding of

an abstraction in the λ-calculus. We have therefore $N' = [\![\lambda b.N'_0]\!]^\lambda$ for some atom b and $N'_0 \in \Lambda$. Now, by Theorem 30, applicative bisimilarity is preserved under reduction and we thus have that $(N_1\ N_2) \simeq \lambda b.N'_0$ and by transitivity of $\simeq$ also that $\lambda b.N'_0 \simeq \lambda a.M'$. Lemma 25 guarantees that the the transition $[\![\lambda b.N'_0]\!]^\lambda \overset{[\![P]\!]^\lambda}{\to} [\![N'_0[P/b]]\!]^\lambda$ is provable, so the second premise of the rule is satisfied and instantiated accordingly. In particular, $N'' = [\![N'_0[P/b]]\!]^\lambda$. Now, since $\lambda a.M' \simeq \lambda b.N'_0$ we have that $M'[P/a] \simeq N'_0[P/b]$. We finally have that $[\![(N_1\ N_2)]\!]^\lambda \overset{[\![P]\!]^\lambda}{\to} [\![N'_0[P/b]]\!]^\lambda$ and $[\![M'[P/a]]\!]^\lambda \mathcal{R} [\![N'_0[P/b]]\!]^\lambda$.

Case 3) is symmetric to case 2) and not shown. We omit the details also for case 4). It is suffice to notice that from two applications M and N we obtain two abstractions after the first reduction step. From this point we find ourselves in case 1) and we can proceed as above.

Completeness: It suffices to show that the relation

$$\mathcal{R} = \{(M, N) \mid [\![M]\!]^\lambda \leftrightarroweq [\![N]\!]^\lambda\}$$

is an applicative bisimulation.

To this end, notice first of all that $\mathcal{R}$ is symmetric, since so is $\leftrightarroweq$. Assume now that $[\![M]\!]^\lambda \leftrightarroweq [\![N]\!]^\lambda$. We have to prove first that if $M \to M'$ then also $N \to N'$ for some N' such that $[\![M']\!]^\lambda \leftrightarroweq [\![N']\!]^\lambda$. By Theorem 17 which states the operational correctness of transitions $\to$, we have that $M \to M'$ only when a transition $[\![M]\!]^\lambda \to M''$ is provable for some M' such that $[\![M'']\!]^\lambda = M'$. Assume this is the case. Since $[\![M]\!]^\lambda \leftrightarroweq [\![N]\!]^\lambda$, we have that also a transition $[\![N]\!]^\lambda \to N''$ is provable, for some N'' such that $M'' \leftrightarroweq N''$. By Lemma 26, which states that a reduction always produce an abstraction, we have that M'' must be an abstraction. By Lemma 27 we know that M'' is the encoding of some λ-term. Since M'' is an abstraction, by a simple inspection over the encoding we can prove that M'' is the encoding of an abstraction in the λ-calculus. We have therefore $M'' = [\![\lambda a.M'_0]\!]^\lambda$, for some atom a and term $M'_0 \in \Lambda$, and by the same reason, $N'' = [\![\lambda b.N'_0]\!]^\lambda$ for some atom b and term $N'_0 \in \Lambda$. Now, since $[\![\lambda a.M'_0]\!]^\lambda \leftrightarroweq [\![\lambda b.N'_0]\!]^\lambda$, they agree also on transitions $\overset{P'}{\to}$ for all terms binding-closed terms P'. We know by Lemma 29 that in this formulation of the lazy λ-calculus that P' can be only a binding-closed term, i.e. $P' \in \mathbb{C}(\Sigma^\lambda)^0$. By Lemma 28, which states that closedness is preserved by the encoding, we also have that every $P' \in \mathbb{C}(\Sigma^\lambda)^0$ is the encoding of a λ-term $P \in \Lambda^0$. We therefore have that the terms $[\![\lambda a.M'_0]\!]^\lambda$ and $[\![\lambda b.N'_0]\!]^\lambda$ agree on transitions of the form $\overset{[\![P]\!]^\lambda}{\to}$ for all λ-terms $P \in \Lambda^0$. Now, by Lemma 25, for all λ-terms $P \in \Lambda^0$ we have that $[\![\lambda a.M'_0]\!]^\lambda \overset{[\![P]\!]^\lambda}{\to} [\![M'_0[P/a]]\!]^\lambda$ and $[\![\lambda b.N'_0]\!]^\lambda \overset{[\![P]\!]^\lambda}{\to} [\![N'_0[P/b]]\!]^\lambda$. Since $[\![\lambda a.M'_0]\!]^\lambda \leftrightarroweq [\![\lambda b.N'_0]\!]^\lambda$, also the targets of these transitions are bisimilar, and we have that $[\![M'_0[P/a]]\!]^\lambda \leftrightarroweq [\![N'_0[P/b]]\!]^\lambda$. We can therefore conclude that $M'_0[P/a] \mathcal{R} N'_0[P/b]$, for all $P \in \Lambda^0$.

Lemma 25 guarantees that there are no other $\overset{[\![P]\!]^\lambda}{\to}$ transitions from $[\![\lambda a.M'_0]\!]^\lambda$ and $[\![\lambda b.N'_0]\!]^\lambda$ that we need to consider. The proof is therefore complete.