Nominal SOS

Matteo Cimini[a] MohamamdReza Mousavi[b] Michel A. Reniers[c]
Murdoch J. Gabbay[d]

[a] *Department of Computer Science, Reykjavík University, Reykjavík, Iceland*

[b] *Department of Computer Science, TU/e, Eindhoven, The Netherlands*

[c] *Department of Mechanical Engineering, TU/e, Eindhoven, The Netherlands*

[d] *Computer Science Department, Heriot-Watt University, 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.

Keywords: SOS, Nominal SOS, Nominal calculi, λ-calculus, π-calculus.

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 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 [24], reprinted in [25], as a logical and structural approach to defining operational semantics.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

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.

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., [3,21] and [6,13,20], respectively. Many programming and specification languages make use of the concepts of names and binders. For example, in the π-calculus [18,19,29], names are first-class objects and the whole language is built on the idea that concurrent agents communicate by exchanging names. Incorporating nominal notions within SOS has received some attention in recent years, but nevertheless the meta-theory of SOS is not sufficiently adapted for these new frameworks. 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 [11,31].

The most important notion of equivalence of programs in the context of SOS is bisimilarity [22]. We argue that this notion, taken as it is, is not satisfactory and we adapt bisimilarity in order to better suit our context with binders. We call this equivalence *nominal bisimilarity*.

Basic notions such as α-conversion and substitution are essential parts of most nominal calculi. We show that these notions can be naturally captured in the Nominal SOS framework. Moreover, we give evidence of the expressiveness of our framework by modeling two of the most prominent examples of nominal calculi, namely the lazy λ-calculus and the early π-calculus. For both we show that our specifications coincide operationally with the original definitions of [2] and [29], respectively. We moreover prove that in the case of the π-calculus our notion of nominal bisimilarity coincides with the well-known open bisimilarity of Sangiorgi [29,28]. Finally, we show that nominal bisimilarity in the context of our formulation of the lazy λ-calculus coincides with the applicative bisimilarity of Abramsky [2].

Proofs are omitted in the main text and the reader can find them, together with a fully elaborated account of Nominal SOS, in [8]. This document extends the last chapter of Cimini's Ph.D. thesis [7].

Structure of the paper. The rest of the paper is organized as follows. In Section 2 we define nominal terms and in Section 3 we define the framework of Nominal SOS. We show in Section 4 how α-conversion and different types of substitution can be accommodated in Nominal SOS. Section 5.1 is devoted to our formulation of the π-calculus and Section 5.2 addresses the lazy λ-calculus. In Section 6 we discuss related work and Section 7 concludes the paper.

2 Nominal terms

The following definitions of *sorts* and *nominal signature* are familiar from [31].

Definition 2.1 (Sorts) *Sorts are defined inductively by the following grammar:*

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

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

Intuitively, $[\mathbb{A}]\sigma$ is a sort whose elements are functions from objects of sort $\mathbb{A}$ 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 2.2 (Nominal signature) *A* nominal signature Σ *is a triple* (Δ, A, F)*, where*

(i) Δ *is a set of* base sorts *ranged over by* δ*,*

(ii) A *is a set of* atom sorts *ranged over by* $\mathbb{A}$*, and*

(iii) 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* $n \geq 0$*.*

For each atom sort $\mathbb{A}$, we fix a countably infinite set of *atoms* $a_{\mathbb{A}}$, $b_{\mathbb{A}}$, $c_{\mathbb{A}}$, $d_{\mathbb{A}}$ ..., and for each sort σ, we assume a countably infinite set V_σ of *variable symbols* x_σ, y_σ, z_σ .... We sometimes write just f, a, b, c, d, n, m, and x, y, z, leaving arities and sorts implicit (but still present). We assume that all these sets of symbols are pairwise disjoint.

Definition 2.3 (Nominal terms) *Given a signature* $\Sigma = (\Delta, A, F)$*, 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*:*

$$t ::= x_\sigma \mid a_{\mathbb{A}} \mid ([a_{\mathbb{A}}]t_\sigma)_{[\mathbb{A}]\sigma} \mid (f_{(\sigma_1 \times \ldots \times \sigma_n) \to \delta}(t_{\sigma_1}, \ldots, t_{\sigma_n}))_\delta$$

where $\mathbb{A} \in A$*,* $a_{\mathbb{A}} \in \mathbb{A}$*,* $x_\sigma \in V_\sigma$ *and* $f \in F$ *with arity* $(\sigma_1 \times \ldots \times \sigma_n) \to \delta$*.*

The subscripts of nominal terms control sorting and we tend to omit them when they are clear from the context or immaterial. We call $[a]t$ an *abstraction*. We have not introduced any means to introduce pairs of terms, as we only use product sorts to give a sort to transition relations (see Section 3).

For a nominal term t, the following definitions will be useful in the remainder of the paper. In what follows, f and g are a unary and a binary function symbol.

- $\mathcal{A}(t)$ stands for the set of atoms that occur in t. For example, $\mathcal{A}(f([a]g(a, b))) = \{a, b\}$.

- $\mathrm{ba}(t)$ 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, $\mathrm{ba}(f([a]g(a,b))) = \{a\}$.
- $\mathrm{fa}(t)$ 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 $\mathrm{fa}(t)$ the set of *free* atoms of t. For example, $\mathrm{fa}(f([a]g(a,b))) = \{b\}$ and also $\mathrm{fa}(g(f([a]a), a) = \{a\}$.
- An atom a is *fresh* in t whenever $a \notin \mathrm{fa}(t)$. We also say that a term t is *binding-closed* if $\mathrm{fa}(t) = \emptyset$, i.e., the term t does not contain free atoms. [1]

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.

3 Nominal SOS

Suppose a is an atom and t is a term of some sort. We call a formula $a\#t$ a *freshness assertion*. In what follows we give a derivation system in order to derive freshness assertions.

Definition 3.1 (Freshness derivation rules) *Let Σ be a nominal signature, and let the atom a and the term t be over the signature Σ. We say that $a\#t$ is* derivable *when it may be derived using the following rules, where a and b are distinct atoms.*

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

These derivation rules are familiar from existing work [31,9].

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

Definition 3.2 (Nominal Transition System Specification) *A* nominal transition system specification *(*NTSS*) is a triple (Σ, R, D) consisting of:*

(i) *A nominal signature Σ;*

(ii) *A set of* (transition) relation symbols *R. 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 may 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'.*

[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 [2,4]. We adopt a different nomenclature in order to avoid confusion with the standard concept of closed term of SOS, i.e., a term that contains no variables.

(iii) *A set of derivation rules D (see below).*

Given an NTSS T, we denote with $\mathcal{A}(T)$ the set of atoms of the signature of T. 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 also the unit sort, then r is a predicate symbol. We may silently drop σ_l (and σ') if they are the unit sort.

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 $t \overset{l}{\nrightarrow}_r$ 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.

Definition 3.3 (Derivation rule) *A* derivation rule *is of the form*

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

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,*
- *$\{t_j \overset{l_j}{\nrightarrow}_{r_j} \mid j \in J\}$ 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, called 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. An NTSS is *positive* when all its deduction rules are positive. Positive NTSS's come with a natural notion of semantics, i.e., the set of provable closed transitions formulae; the same notion is adopted for the semantics of freshness formulae by means of the derivation rules given in Definition 3.1. In this paper we restrict ourselves to positive NTSS's; the semantics of full Nominal SOS can be found in [7, Section 5.3.1].

The most important notion of equivalence between programs defined in SOS is bisimilarity [17,22]. Unfortunately, this equivalence turns out not to be satisfactory in a context with binders. This point is carefully explained, in the context of the π-calculus, on pages 64-65 of [29], where it is shown that

the two processes $P = \nu z.\overline{x}z$ and $Q = \nu z.(\overline{x}z \mid\mid \nu w.\overline{w}y)$ are distinguished since P can perform a transition $\xrightarrow{\overline{x}(y)}$ while Q is not able to perform any $\xrightarrow{\overline{x}(y)}$ transitions; the reason is that Q is not able to change its bound variable to y since y is not fresh in Q.

Nominal bisimilarity is thus introduced below as an adaptation of the ordinary bisimilarity that is aware of binding. What happens in the theory of the π-calculus is that bisimilarity is adjusted and transitions of the form $\xrightarrow{x(z)}$ are matched only for those bound variables that are fresh in both terms. Our notion of bisimilarity revisits the ordinary bisimilarity in such a manner. In the remainder of the paper we will relate nominal bisimilarity to important notions of equivalence in the literature.

Definition 3.4 (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 for all closed terms P and Q and labels l such that $P \sim Q$ and $a\#P$ and $a\#Q$ for all $a \in \mathrm{ba}(l)$, it holds that if $P \xrightarrow{l} P'$ then there exists Q' such that $Q \xrightarrow{l} Q'$ and $P' \sim Q'$.*

4 Substitution and α-conversion

4.1 Substitution transitions

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 in a uniform fashion within the framework of Nominal SOS.

Term-for-atom substitutions are typically employed by higher-order calculi, such as the λ-calculus, the Calculus of Higher Order Communicating Systems (CHOCS) [30] and the Higher-Order π-calculus [26], just to mention a few. Given a nominal signature, we proceed to generate the following types of deduction rules with the goal of proving transitions of the form $t_1 \xrightarrow{a \overset{T}{\mapsto} t_2} t_3$ for some atom a and terms t_1, t_2 and t_3. This type of transition should be read as *the term t_2 replaces the atom a in the term t_1 leading to the term t_3*. For all atoms a and function symbols f, we have the following rules.

$$a \xrightarrow{a \overset{T}{\mapsto} z} z \ (\mathrm{a1_{Ts}}) \qquad \frac{a\#x}{a \xrightarrow{x \overset{T}{\mapsto} z} a} \ (\mathrm{a2_{Ts}}) \qquad \frac{x \xrightarrow{y \overset{T}{\mapsto} z} x' \quad a\#z \quad a\#y}{[a]x \xrightarrow{y \overset{T}{\mapsto} z} [a]x'} \ (\mathrm{abs1_{Ts}})$$

$$[a]x \xrightarrow{a \overset{T}{\mapsto} z} [a]x \ (\mathrm{abs2_{Ts}}) \qquad \frac{\left\{ x_i \xrightarrow{y \overset{T}{\mapsto} z} x_i' \mid 0 < i \leq n \right\}}{f(x_1, x_2, \ldots, x_n) \xrightarrow{y \overset{T}{\mapsto} z} f(x_1', x_2', \ldots, x_n')} \ (\mathrm{f_{Ts}})$$

The reader can infer the sort of the variables used in the rules by their usage. In the rules above we use variables of atom sort whenever possible with the exception of rule ($a1_{\mathrm{Ts}}$) that would appear less readable. The reader may be more familiar with the syntactic substitution operation, defined below, where M and N are closed terms and a and b are distinct atoms.

$$a[N/a] = N \qquad a[N/b] = a \qquad ([a]M)[N/a] = [a]M$$
$$([a]M)[N/b] = [a](M[N/b]) \quad \text{if } a \text{ is fresh in } N$$
$$f(M_1, M_2, \ldots, M_n)[N/a] = f(M_1[N/a], M_2[N/a], \ldots, M_n[N/a])$$

The following theorem states that the two notions (substitution transitions and syntactic substitutions) correspond [2] .

Theorem 4.1 (Correctness of Substitution Transitions) *Let T be an NTSS. Let M and N be closed terms, and a be an atom. Then, it holds that $M \xrightarrow{a \overset{T}{\mapsto} N} M'$ if and only if $M' = M[N/a]$.*

Atom-for-atom substitution is used in calculi such as the π-calculus [29,19] and its variants. The same set of rules provided for the term-for-atom substitution are able to model the atom-for-atom substitution. These transitions are denoted $\xrightarrow{x \overset{A}{\mapsto} z}$ and the first and second argument of this label range over atoms.

A syntactic atom-for-atom substitution over nominal terms, together with its corresponding correctness theorem, can be provided. It turns out to be just a straightforward adaptation of Theorem 4.1.

4.2 α-conversion Transitions

The notion of α-conversion is a natural equivalence guaranteeing that the exact atom chosen in binders is not important and can be indeed replaced by any other atom (while avoiding capture). Thanks to freshness assertions, we can accommodate α-conversion in our framework as an ordinary transition relation. Given a nominal signature, the following deduction rules define $\to_\alpha$. For all atoms a and b and function symbols f, we have the following rules. [3]

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

[2] Theorem 4.1 is stated as Theorem 2 in Section 4.1 of [8] and proved in Section 13 of that paper.

[3] In the nominal world, the standard definition of α-equivalence is based on permutations, see [11] for instance. However, we preferred to model the standard definition of α-equivalence.

$$\frac{\{x_i \to_\alpha x_i' \mid 0 < i \leq n\}}{f(x_1, x_2, \ldots, x_n) \to_\alpha f(x_1', x_2', \ldots, x_n')}\ (\mathrm{f}_\alpha) \qquad \frac{x \to_\alpha y \quad y \to_\alpha z}{x \to_\alpha z}\ (\alpha \cdot \mathrm{upTo}\alpha)$$

The reader will notice that α-conversion transitions rely on those for the atom-for-atom substitution (rule $(abs1_\alpha)$). Throughout the paper, whenever we say that the rules above for α-conversion transitions are present in an NTSS, this implies that also the rules for atom-for-atom substitutions are present.

The reader is perhaps familiar with the syntactic version of α-conversion, defined below.

Definition 4.2 (α-conversion over nominal terms) *Let T be an NTSS. The relation $=_\alpha$ is the least congruence on nominal terms over the signature of T, such that, for all closed term M and an atom b, if b is fresh in M then $[a]M =_\alpha [b]M[b/a]$.*

The set of rules for α-conversion transitions generated above actually behaves according to the syntactic α-conversion, as stated in the following theorem [4].

Theorem 4.3 (Correctness of α-conversion transitions) *Let T be an NTSS. For all closed terms M and N over the signature of T, it holds that $M \to_\alpha N$ if and only if $M =_\alpha N$.*

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 by augmenting the NTSS with a deduction rule, given below.

Definition 4.4 (Transitions up to α-equivalence) *Let T be an NTSS and l be a label of 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 Section 4.1, and the deduction rule:*

$$\frac{x \to_\alpha y \quad y \xrightarrow{l} z}{x \xrightarrow{l} z}\ (l \cdot \mathrm{upTo}\alpha).$$

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

[4] Theorem 4.3 is stated as Theorem 3 in Section 4.2 of [8] and proved in Section 14 of that paper.

5 Examples

In this section we provide some evidence of the expressiveness, and perhaps naturalness, of Nominal SOS by formulating in our framework two classical calculi, namely the early π-calculus [29,19] and the lazy λ-calculus [2].

5.1 The early π-calculus and open bisimilarity

The signature Σ^π of our π-calculus is modelled by a base sort P and atom sort C (for processes and channels, respectively) and the following function symbols.

(i) $\mathbf{0} :\to P$ for inaction (deadlock),
(ii) $\tau._- : P \to P$ for τ-prefix,
(iii) $out(\_,\_,\_) : (C \times C \times P) \to P$ for output prefix,
(iv) $in(\_,\_) : (C \times [C]P) \to P$ for input prefix,
(v) $\nu(\_) : [C]P \to P$ for restriction,
(vi) $\_ || \_ : (P \times P) \to P$ for parallel composition,
(vii) $\_ + \_ : (P \times P) \to P$ for nondeterministic choice,
(viii) $!\_ : P \to P$ for parallel replication.

The syntax employed for input and output prefixes differs slightly from the standard notation used in the π-calculus. In particular, our term $out(a, b, P)$ corresponds to the process $\overline{a}b.P$ of the π-calculus, and $in(a, [b]P)$ corresponds to $a(b).P$. The same choice is adopted for the labels.

Below, we specify the semantics of the early π-calculus in Nominal SOS. Since in our framework labels are open terms, we display an input transition label as $in(a, b)$, assuming a different operator in accepting two atoms as arguments. For presentational purposes, we use the same names to stipulate the meaning of the transitions. For the same reasons, we model an output transition label as $out(a, b)$ and a bound output transition label as $bout(a, [b]\mathbf{0})$, abbreviated as $bout(a, [b])$ throughout the text. The set of rules of the signature Σ^π contains the following rules, we use α to range over labels and a, b and c to range over atoms.

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

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

$$\frac{x_1 \xrightarrow{bout(a,[b])} y_1 \quad b\#x_2}{x_1 \,||\, x_2 \xrightarrow{bout(a,[b])} y_1 \,||\, x_2}\ (\text{parRes1}) \qquad \frac{x_1 \xrightarrow{out(a,b)} y_1 \quad x_2 \xrightarrow{in(a,b)} y_2}{x_1 \,||\, x_2 \xrightarrow{\tau} y_1 \,||\, y_2}\ (\text{com1})$$

$$\frac{x_1 \xrightarrow{bout(a,[b])} y_1 \quad x_2 \xrightarrow{in(a,b)} y_2 \quad b\#x_2}{x_1 \,||\, x_2 \xrightarrow{\tau} \nu([b](y_1 \,||\, y_2))}\ (\text{close1}) \qquad \frac{x \xrightarrow{\alpha} y}{!x \xrightarrow{\alpha} y \,||\, !x}\ (\text{repl})$$

$$\frac{x \xrightarrow{out(z,a)} y \quad a \neq z}{\nu([a]x) \xrightarrow{bout(z,[a])} y}\ (\text{open}) \qquad c \notin \mathrm{ba}(\alpha) \quad \frac{x \xrightarrow{\alpha} y \quad c\#\alpha}{\nu([c]x) \xrightarrow{\alpha} \nu([c]y)}\ (\text{res})$$

For the sake of brevity, we omit the symmetric versions of rules (sum1), (par1), (parRes1), (com1) and (close1). Moreover, for each label l, $\xrightarrow{l}$ is up to α-equivalence. Following the recipe of Definition 4.4 we add to our NTSS the set of rules for α-conversion transitions and the rules for atom-for-atom substitution transitions. We set the atom-for-atom transition relations to be up to α-equivalence, too.

The reader must notice that the complicated side-conditions of the ordinary formulation of π-calculus are here replaced by rather simpler freshness conditions, see rules (parRes1) and (close1).

We denote by Π the set of π-terms of [29]. The encoding $[\![\cdot]\!]^\pi$ is a map from Π into terms of our nominal π-calculus. The mapping is straightforward and its definition is omitted here; it can however be found in [8]. [5] The following theorem establishes that our formulation of the early π-calculus is operationally correct with respect to its original formulation [6].

Theorem 5.1 (Operational Correspondence: early π-calculus) *For all $P, Q \in \Pi$, $P \xrightarrow{\alpha} Q \Leftrightarrow [\![P]\!]^\pi \xrightarrow{[\![\alpha]\!]^\pi} [\![Q]\!]^\pi$, where α ranges over the labels of the form τ, ab, $\overline{a}b$ and $\overline{a}(b)$ from the original early π-calculus.*

The reader may wonder what is the equivalence over π-calculus terms that corresponds to nominal bisimilarity. 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 [29, Section 4.2] and [28].

Definition 5.2 (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 (see Definition 1.1.3 on page 14 of [29]), 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 [29] and [28]. In

[5] The encoding can be found in Section 5.2 of [8]. By way of example the reader can consider that $[\![\nu z.\overline{x}(z)||z(a).0]\!]^\pi = \nu([z](bout(x,z)||in(z,[a]0)))$.

[6] Theorem 5.1 is stated as Theorem 5 in Section 5.2 of [8] and proved in Section 16 of that paper.

Definition 5.2, it is important to note that the ranging over all the substitutions is performed at each step of the bisimulation game.

Theorem 5.3 (Open bisimilarity and Bisimilarity coincide) *For all $P, Q \in \Pi$, $P \underline{\leftrightarrow}^o Q$ if, and only if, $[\![P]\!]^\pi \leftrightarrow [\![Q]\!]^\pi$.*

Theorem 5.3 essentially holds because in our nominal formulation of the π-calculus, nominal bisimilarity also takes into account the substitutions transitions [7]. In [7, Section 5.21], it is shown that if nominal bisimilarity is adapted not to match the substitution transitions it would coincide with the ordinary bisimilarity over the π-calculus.

5.2 The lazy λ-Calculus and applicative bisimilarity

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.

(i) $(\_) : A \to L$: A unary function symbol for creating terms from atoms;

(ii) $\lambda(\_) : [A]L \to L$: A unary function symbol for abstractions;

(iii) $\_\ \_ : (L \times L) \to L$: A binary function symbol for application.

The semantics includes a reduction transition $\to$, here displayed with no label to remain in line with the standard notation from [2], transitions $\xrightarrow{P}$ for terms P of sort L, and the rules for term-for-atom substitution transitions as generated in Section 4.1, but where rules $(a1_{\mathrm{Ts}})$ and $(a2_{\mathrm{Ts}})$ are modified to deal with embedding of atoms. For instance, rule $(a1_{\mathrm{Ts}})$ becomes $(a) \xrightarrow{a \overset{T}{\mapsto} z} z$.

The set of rules of the signature Σ^λ contains the following derivation rules, which define the operational semantics of our version of the lazy λ-calculus, for all atoms a.

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

$$\frac{x_0 \to y_0 \quad y_0 \xrightarrow{x_1} y_1 \quad y_1 \to y_2}{(x_0\ x_1) \to y_2}\ (\text{app1AP}) \qquad \frac{(x_0\ x_1) \to y_1 \quad y_1 \xrightarrow{x_2} y_2}{(x_0\ x_1) \xrightarrow{x_2} y_2}\ (\text{app2AP})$$

Moreover, the transition relations $\to$, $\xrightarrow{P}$ for any binding-closed term P [8], and the term-for-atom substitution transitions are up to α-equivalence. Recall

[7] Theorem 5.3 is stated as Theorem 7 in Section 6 of [8] and proved in Section 22 of that paper.

[8] The premises $\forall b.(b\#y)$ of rule (abs2AP) ensure that the parameter passing is performed with binding-closed terms only. This characterization already appeared in [9, Section 9.2].

that, by Definition 4.4, this means that the set of rules of Σ^{λ} contains also the rules for α-conversion transitions and the rules for atom-for-atom substitution transitions. Rules $(a1_{\mathrm{Ts}})$ and $(a2_{\mathrm{Ts}})$ are modified as before also for atom-for-atom substitution transitions and these transitions are set to be up to α-equivalence.

We denote by Λ the set of λ-terms of [2,4], and by Λ^0 the set of those that do not contain free variables. The encoding $[\![\cdot]\!]^{\lambda}$ is a map from Λ into terms of our nominal λ-calculus. The mapping is straightforward and not presented here; it can however be found in [8]. [9] The following theorem establishes the operational correctness of our formulation of the lazy λ-calculus with respect to its original formulation for λ-terms in Λ^0 [10] .

Theorem 5.4 (Operational Correspondence: lazy λ-calculus) *For all $M, N \in \Lambda^0$, $M \to N \Leftrightarrow [\![M]\!]^{\lambda} \to [\![N]\!]^{\lambda}$.*

The reader should notice that if we ruled out the premises $\forall b.(b\#y)$ from rule (abs2AP) the operational correspondence of Theorem 5.4 would hold for the set of all λ-terms. The reason we restrict the parameter passing to binding-closed terms only is that it ties up directly with the study that follows.

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.

Definition 5.5 (Applicative bisimilarity in the λ-calculus) 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 very unsatisfactory over terms that contain free variables. For instance, for all variables a, b and c, it holds that $a \simeq b$ and $\lambda a.b \simeq \lambda a.c$.

The following theorem states that applicative bisimilarity in the lazy λ-calculus coincides with nominal bisimilarity in the nominal formulation of the lazy λ-calculus given above [11] .

Theorem 5.6 (Applicative and Nominal Bisimilarity coincide) *For all $M, N \in \Lambda^0$, $M \simeq N$ if, and only if, $[\![M]\!]^{\lambda} \leftrightarrows [\![N]\!]^{\lambda}$.*

[9] The encoding can be found in Section 5.1 of [8]. By way of example the reader can consider that $[\![\lambda x.\lambda y.(x\ y)]\!]^{\lambda} = \lambda([x](\lambda([y](x\ y))))$.
[10] Theorem 5.4 is stated and proved as Theorem 18 in Section 25 of [8].
[11] Theorem 5.6 is stated as Theorem 13 in Section 7.1 of [8] and proved in Section 25 of that paper.

6 Related Work

We are aware of a number of existing approaches that accommodate names and binders inside the SOS framework. The frameworks that are most relevant to the work presented in this paper are by Miller and Tiu in [16] ($FO\lambda^{\Delta\nabla}$), by Lakin and Pitts in [15] (MLSOS) and in [14] (αML) and by Gacek, Miller and Nadathur in [12] (Abella).

As a first difference, Nominal SOS is the only approach that directly extends the formal framework of SOS. We identify some benefits supporting this choice. First, users that are familiar with the SOS framework will find Nominal SOS easy to use. Secondly, although a meta-theory of SOS for calculi with binders can be carried out with the frameworks mentioned above, Nominal SOS seems to be close enough to ordinary SOS. In this respect, a meta-theory for binders can follow by and large the same lines of the meta-theory of ordinary SOS, which has been successfully developed for over 20 years, see [21]. We also expect that already existing results from the meta-theory of SOS would lift to Nominal SOS with relatively little effort.

Some technical differences between Nominal SOS and the systems mentioned above are worth a mention. For instance in MLSOS and α-ML, only restricted operations are allowed on atoms and programs do not depend upon concrete atoms. [12] In Nominal SOS, languages can instead be defined in a way that a particular atom may affect the computation. $FO\lambda^{\Delta\nabla}$ and Abella are based on the so-called λ-*tree approach* to syntax where a logic that has its roots in the λ-calculus takes care of the binding management. In Nominal SOS, the management of binders is not delegated to an underlying layer and users need to specify the treatment of binders completely. As another difference, in the mentioned approaches α-conversion is built-in and guaranteed on the meta-level. In Nominal SOS α-conversion is not built-in but it can be automatically generated from the signature. The user can replace it and experiment with alternative notions at will. Moreover, we prefer to model α-equivalence as a transition 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.

We refer the reader to Section 5.8 of [7], where related works are considered in much more detail.

7 Conclusions and Future Work

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, substi-

[12] This property is known in the nominal world as *the equivariance property*, see [11] and especially [23].

tution and α-conversion. We used the framework to specify the semantics of the early π-calculus and lazy λ-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. We also proved that nominal bisimilarity coincides with Abramsky's applicative bisimilarity in the context of the lazy λ-calculus.

The main goals of our future work are 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 [27], the psi-calculi [5] and the object calculi [1]. Also, we plan to address different notions of equivalence betweens terms. To begin with, we plan to adapt nominal bisimilarity in order for it to coincide with open bisimilarity with distinctions [29,28], when applied to π-calculus terms. Our main goal is however to develop the meta-theory of Nominal SOS. By way of example, it would be worth providing congruence formats for behavioural semantics, possibly generalizing those proposed in [32] and [10], for instance.

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] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics, Revised edition*. North Holland, 1984.

[5] 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.

[6] 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.

[7] Matteo Cimini. *Contributions to the Meta-Theory of Structural Operational Semantics*. Ph.D. in Computer science, School of Computer Science, Reykjavik University, 2012. available at http://cimini.info.

[8] Matteo Cimini, Mohammadreza Mousavi, Michel A. Reniers, and Murdoch J. Gabbay. Nominal SOS - Online Resource. available at: http://cimini.info/publications/publications.html.

[9] Maribel Fernandez and Murdoch J. Gabbay. Nominal rewriting. *Information and Computation*, 205(6):917–965, 2007.

[10] 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.

[11] 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.

[12] 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.

[13] Pieter H. Hartel. Letos - a lightweight execution tool for operational semantics. *Software - Practice and Experience*, 29:1379–1416, December 1999.

[14] Matthew R. Lakin. *An executable meta-language for inductive definitions with binders*. PhD thesis, University of Cambridge, 2010.

[15] Matthew R. Lakin and Andrew M. Pitts. A metalanguage for structural operational semantics. In *Symposium on Trends in Functional Programming*, pages 1–16, 2007.

[16] Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic*, 6:749–783, October 2005.

[17] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[18] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part i. *Information and Computation*, 100, 1989.

[19] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, ii. *Information and Computation*, 100:41–77, September 1992.

[20] Mohammad Reza Mousavi and Michel A. Reniers. Prototyping sos meta-theory in maude. *Electronic Notes in Theoretical Computer Science*, 156:135–150, May 2006.

[21] 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.

[22] David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, London, UK, 1981. Springer-Verlag.

[23] Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:2003, 2002.

[24] Gordon D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[25] Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[26] Davide Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996.

[27] 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.

[28] Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33(1):69–97, 1996.

[29] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge Univ. Press, 2001.

[30] Bent Thomsen. A theory of higher order communicating systems. *Information and Computation*, 116(1):38–57, 1995.

[31] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323:473–497, 2004.

[32] 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.