Translating Specifications from Nominal Logic to CIC with the Theory of Contexts *

Marino Miculan Ivan Scagnetto Furio Honsell

Dept. of Mathematics and Computer Science, University of Udine
Via delle Scienze 206, I-33100 Udine, Italy.
{miculan,scagnett,honsell}@dimi.uniud.it

Abstract

We study the relation between Nominal Logic and the Theory of Contexts, two approaches for specifying and reasoning about datatypes with binders. We consider a natural-deduction style proof system for intuitionistic nominal logic, called NINL, inspired by a sequent proof system recently proposed by J. Cheney. We present a translation of terms, formulas and judgments of NINL, into terms and propositions of CIC, via a weak HOAS encoding. It turns out that the (translation of the) axioms and rules of NINL are derivable in CIC extended with the Theory of Contexts (CIC/ToC), and that in the latter we can prove also sequents which are not derivable in NINL. Thus, CIC/ToC can be seen as a strict extension of NINL.

Categories and Subject Descriptors D.3.1 [*Formal Definitions and Theory*]: Syntax; F.3.1 [*Specifying and Reasoning about Programs*]: Specification techniques; F.4.1 [*Mathematical Logic*]: Lambda calculus and related systems; Mechanical theorem proving

General Terms Theory, Languages, Verification.

Keywords Languages with binders, Nominal Logics, Calculus of Inductive Constructions, Theory of Contexts, logical expressivity.

1. Introduction

In recent years, many different logics for reasoning about languages with binders have been proposed; see e.g. [4, 9, 3, 8] among others. All these logics can be used as "metalogical specification systems": a generic object system (with binders) can be represented faithfully in these logics ("encoded"), and then the logic provides tools and techniques for reasoning about the object system. In spite of this common aim, these logics differ in many aspects: the kind of the logic (first-order, higher-order, type theory), the way binders are represented (first-order, second-order, higher-order), the "intended behavior" of the bound symbols (variables, names, linear names, ordered names, ...), etc. Thus, a given object system can be encoded in different logics, using different techniques, yielding quite different metalogical systems for reasoning about it.

* Work supported by EU CA FP6-IST-510996 "TYPES".

MERλIN'05 September 30, 2005, Tallinn, Estonia.

A natural question which arises at this point is: how to compare these logics for binders? A possible way is to check whether, for a given object system $\mathcal{S}$, all properties which can be proved in the logic $\mathcal{L}_1(\mathcal{S})$ can be proved also in the logic $\mathcal{L}_2(\mathcal{S})$. More precisely, this "expressiveness" comparison can be carried out by defining a *translation* of the formulas of one system into the other's, and checking that the translation of properties derivable in $\mathcal{L}_1(\mathcal{S})$ are still derivable in $\mathcal{L}_2(\mathcal{S})$. An example of this kind of translation is in [1], where a conservative translation of Miller and Tiu's $FO\lambda^{\nabla}$ [8] into Nominal Logics is presented.

This is the subject of this paper. We describe a translation of *Nominal Logic* [9] into the Calculus of Inductive Constructions extended with the Theory of Contexts [4, 5].

In our opinion, this comparison is important for many reasons. First, we can check if the target logic (CIC/ToC in this case) is "expressive enough" to cover all the properties derivable in the source logic (Nominal Logic). In second place, the translation allows to enlighten the similarities and differences between different logics. Third, the translation will streamline an encoding methodology of object systems in CIC/ToC. Moreover, since CIC/ToC is easily implemented taking advantage of existing machine-assisted infrastructure (namely Coq [5, 6]), the translation can be used for implementing NL specifications (albeit this implementation would not be as "efficient" and full-fledged as specifically designed proof assistants/theorem provers). Finally, a translation of Nominal Logic into CIC/ToC can enlighten also the connections between $FO\lambda^{\nabla}$ and CIC/ToC, in virtue of the translation given in [1].

Notice that we *do not* advocate a "reductionist" approach, aiming to single out the "best" logic in which all the others can be embedded–and thus discarded. Our opinion is that there is no "best logic"; instead, the choice of which logic is "best" to reason about a given object system is a trade-off, which depends ultimately on how this logic would be used afterwords. Since these logics are often intended to be used in semi-automated proof assistants or theorem provers, a complete comparison should take into account many aspects beside logical expressiveness. For instance, what can and can't be proved about an object system depends also on the encoding methodology. From a pragmatic point of view, it is important to judge how close reasoning in a particular logic is to informal reasoning. Some other interesting aspects to consider are proof theory, proof search, decidability, model theory, etc.

A such complete comparison between Nominal Logic and CIC/ToC is out of the scope of this paper. Here, we focus on the specific aspect of logic expressivity only.

In fact, the translation of terms and formulas from Nominal Logics to CIC/ToCis not trivial, due to the quite different approaches adopted by the two systems. The first main difference is in the treatment of terms with bound variables: as equivalence classes of first-class terms in nominal logics, or as functional terms

in CIC/ToC according to the HOAS paradigm. Thus, a is free in $\langle a\rangle t$ in NL, while it is not free in $\lambda a.t$ in CIC. As a consequence, in nominal logics all constructors (also the binders) are represented by first-order (i.e., non binding) constants; on the other hand, in CIC/ToC they are represented by truly binding constructors.

Moreover, it is known that the Axiom of Unique Choice is consistent with Nominal Logic, while it is not with CIC/ToC. As a consequence, some functions allowed in Nominal Logic, are not available as functions in CIC/ToC but still are available as relations. This means that the theory of a CIC/ToC specifications is forced to have a relational rather than functional feel. The translation is tricky because it has to deal with all these mismatchings.

Nevertheless, we will show that the translation preserves the derivability of properties, i.e., given a nominal signature, all properties which are derivable in NINL are still derivable in CIC/ToC. The converse is not true: there are properties which can be proved in CIC/ToC but not in NINL. This is not a big surprise, since NINL is a first-order logic while CIC/ToC is higher-order.

Synopsis. We proceed as follows. In Section 2 we present the notion of *nominal signatures*, that is signatures for defining (typed) languages with name binders. In Section 3 we present NINL, a proof system for Nominal Logic over a given nominal signature. Similarly, Section 4 describes the encoding of a given nominal signature in CIC/ToC, using the weak HOAS techniques. The translation from specifications in Nominal Logic to CIC/ToC is given in Section 5. In Section 6 we discuss the properties of this translation. Conclusions and final remarks are in Section 7.

2. Nominal signatures

In this section we present the notion of *nominal signatures*, which are intended to be a way for describing languages with name binders, similarly to [4]. Notice that nominal signatures are different from *binding signatures* [2], because the latter are used for describing languages with binders of *variables* instead of names.

Definition 1 (Nominal signatures) *A* nominal signature $\mathcal{S}$ *is a tuple* $\mathcal{S} = (N, D, C, P)$ *where*

- $N = \{\nu_1, \ldots, \nu_n\}$ *is a collection of* name types symbols*, ranged over by* ν*;*
- $D = \{\delta_1, \ldots, \delta_m\}$ *is a collection of* data types symbols*, ranged over by* δ*. The* sorts σ *and* arities α *are then defined by the following grammar:*

$$\sigma ::= () \mid \nu, \sigma \mid \langle \nu_1 \ldots \nu_k\rangle\delta, \sigma \quad (k \geq 0)$$
$$\alpha ::= \sigma \to \delta$$

- $C = \{c_1{:}\alpha_1, \ldots, c_j{:}\alpha_j\}$ *is a collection of* data constructors*, where* α_i*'s are arities*
- $P = \{p_1{:}\sigma_1, \ldots, p_k{:}\sigma_k\}$ *is a collection of* (atomic) predicate symbols*, where* σ_i*'s are sorts.*

For sake of readibility, we write arities $() \to \delta$ simply as δ.

We say that an arity $(\langle\vec{\nu}_1\rangle\delta_1, \ldots, \langle\vec{\nu}_n\rangle\delta_n) \to \delta$ is a *binding arity* if $|\vec{\nu}_i| > 0$ for some i. Constructors of binding arities are called *binders*. Notice that constructors and predicates are allowed to take names as arguments, to bind names over datatypes, but not to bind names over name types; for instance, a constructor of arity $\langle\nu\rangle\nu \to \delta$ is not allowed.

The intuition behind nominal signatures is the following:

- name types are the sorts where variables, names, etc., are taken from; these types do not have constructors, because variables, names, etc, have no structure.
- Data types are the sorts inductively defined by a set of constructors. The arguments of these constructors may be names, terms, or terms abstracted over names, but cannot be terms abstracted over terms.
- Predicate symbols are intended to represent relations between terms, possibly abstracted over names.

This intuition is made precise in the following definition:

Definition 2 (Language of a nominal signature) *Let* $\mathcal{S}$ *be a nominal signature* $\mathcal{S} = (N, D, C, P)$*. A* typing base *for* $\mathcal{S}$ *is a list* $\Gamma = n_1{:}\nu_1, \ldots, n_k{:}\nu_k$ *where the* n_i *are distinct abstract symbols.*[1]

The language generated by $\mathcal{S}$ *is the least set of (typed) terms, taken up-to* α*-conversion of bound names, which is closed under the following rules:*

$$\frac{(n{:}\nu) \in \Gamma}{\Gamma \vdash n : \nu}\ Start$$

$$\frac{\Gamma, \vec{n}_1 : \vec{\nu}_1 \vdash t_1 : \delta_1 \quad \ldots \quad \Gamma, \vec{n}_k : \vec{\nu}_k \vdash t_k : \delta_k}{\Gamma \vdash c((\vec{n}_1)t_1, \ldots, (\vec{n}_k)t_k) : \delta}\ Constr_c$$

where $c{:}(\langle\vec{\nu}_1\rangle\delta_1, \ldots, \langle\vec{\nu}_k\rangle\delta_k) \to \delta \in C$

$$\frac{\Gamma, \vec{n}_1 : \vec{\nu}_1 \vdash t_1 : \delta_1 \quad \ldots \quad \Gamma, \vec{n}_k : \vec{\nu}_k \vdash t_k : \delta_k}{\Gamma \vdash p((\vec{n}_1)t_1, \ldots, (\vec{n}_k)t_k)\ wf}\ Prop_p$$

where $p{:}(\langle\vec{\nu}_1\rangle\delta_1, \ldots, \langle\vec{\nu}_k\rangle\delta_k) \in P$*.*

Many systems can be specified by nominal signatures. For example, the signature of untyped λ-calculus is

$$\mathcal{S}_\lambda = (\{\nu\}, \{\Lambda\}, \{var{:}\nu \to \Lambda, \lambda{:}\langle\nu\rangle\Lambda \to \Lambda, app{:}(\Lambda, \Lambda) \to \Lambda\}, \{\longrightarrow{:}\ (\Lambda, \Lambda)\})$$

For instance, $\lambda((x)app(var(x), var(x)))$ is the formal notation for the usual $\lambda x.(x\ x)$.

Another interesting example is the signature for the π-calculus, which has three datatypes (processes, free actions and bound actions) and two transition relations [7, 5]:

$$\begin{aligned}\mathcal{S}_\pi = (&\{\nu\}, \{\Pi, FA, BA\},\\ &\{0{:}\Pi, |{:}(\Pi, \Pi) \to \Pi, .{:}(FA, \Pi) \to \Pi, .{:}(BA, \langle\nu\rangle\Pi) \to \Pi,\\ &\ = {:}(\nu, \nu, \Pi) \to \Pi, \nu{:}(\langle\nu\rangle\Pi) \to \Pi, !{:}\Pi \to \Pi\\ &\ in{:}\nu \to BA, out{:}(\nu, \nu) \to FA, \tau{:}FA\},\\ &\{\longrightarrow{:}\ (\Pi, FA, \Pi), \longrightarrow'{:}\ (\Pi, BA, \langle\nu\rangle\Pi)\}).\end{aligned}$$

Therefore, in this signature the term $.(in(n), (x){=}(n, x, 0))$ is the formal notation for the process $n(x).[n = x]0$.

We refer the reader to [4] for further examples.

3. Nominal signatures in Nominal Logic

Given a nominal signature $\mathcal{S}$, we can readily define a nominal logic for reasoning about its properties. In this section we present NINL, yet another proof system for (Intuitionistic) Nominal Logic.

Nominal Logic is a first order logic designed for reasoning about data containing abstract symbols ("names", "variables", etc), *up-to* α*-conversion* of bound symbols. The logic features a primitive operation of *name swapping*, and it is required that all properties are preserved under permutation of bound symbols, i.e., they are *equivariant*. A complete discussion of NINL is out of the scope of this paper; we refer the reader to [9] for a detailed presentation.

The version we present, NINL, is directly inspired to the sequent-style system $\text{NL}^{\Rightarrow}$ presented in [1]; in fact, it can be seen as the same system, turned in Natural Deduction style. This style of presentation is closer to type theory and hence easier to compare with the logic of CIC.

[1] One may think of each n_i as a different number, or string, etc.

3.1 Syntax of terms and formulas

$\mathsf{NINL}(\mathcal{S})$ is a first-order logic with equality over a simply-typed λ-calculus with constants, parametric in a given nominal signature $\mathcal{S}$. (We drop the "$(\mathcal{S})$" when clear from the context). We recall briefly the notation and syntax, following [1] with the difference that we do not introduce an explicit type o of formulas.

For a given signature $\mathcal{S} = (N, D, C, P)$, the types τ of $\mathsf{NINL}(\mathcal{S})$ are defined as follows

$$\tau ::= \delta \mid \nu \mid \tau \to \tau' \mid \langle\nu\rangle\tau$$

where δ ranges over D, and ν over N. Thus, an arity of the form $(\langle\vec{\nu}_1\rangle\delta_1, \ldots, \langle\vec{\nu}_k\rangle\delta_k) \to \delta$ is represented in currified form by the type $\langle\vec{\nu}_1\rangle\delta_1 \to \cdots \to \langle\vec{\nu}_k\rangle\delta_k \to \delta$.

Next, the terms t are defined as follows:

$$t, u ::= x \mid \mathsf{a} \mid \lambda x{:}\tau.t \mid t\ u \mid c \mid \mathit{swap}_{\nu\tau} \mid \mathit{abs}_{\nu\tau}$$

where variable symbols x, y are taken from a countably infinite set V, and name symbols a, b from a disjoint countably infinite set A of *names*. The constants c are the symbols in C; moreover, for each ν, τ there are two constants

$$\mathit{swap}_{\nu\tau} : \nu \to \nu \to \tau \to \tau \qquad \mathit{abs}_{\nu\tau} : \nu \to \tau \to \langle\nu\rangle\tau$$

We will use the letters a, b to denote terms of some name-sort ν (i.e., they are metavariables for names); on the other hand, a, b are atomic constants. The notations $(a\ b) \cdot v$ and $\langle a\rangle u$ are syntactic sugar for the terms $(\mathit{swap}\ a\ b\ v)$, and $(\mathit{abs}\ a\ u)$, respectively. Intuitively, $(a\ b) \cdot t$ represents the term obtained by swapping all occurences of a and b in t; $\langle a\rangle t$ represents the term obtained by "abstracting" a in t, or better, the equivalence class of terms which can be obtained replacing a in t with any fresh name. It is important to notice that the only binder is λ, i.e., constants (and in particular *abs*) *are not* binders. $FV(t)$, $FN(t)$, $FVN(t)$ denote the sets of free variables, name-symbols, or both variables and name-symbols of term t. As usual, terms are taken up to $\alpha\beta\eta$-equivalence.

The *(typing) contexts* are defined as follows:

$$\Sigma ::= \langle\rangle \mid \Sigma, x{:}\tau \mid \Sigma\#a{:}\nu$$

Intuitively, $\Sigma\#a{:}\nu$ declares a variable a ranging over names which do not "interfere" with those in Σ. For instance, $a{:}\nu\#b{:}\nu$ declares two *distinct* "names", or better, two variables (of type ν) which cannot be instantiated with the same name. Similarly, $x{:}\tau\#a{:}\nu$ means that for any t which can be assigned to x, a must be considered as a name not occurring free in t. Therefore, contexts contain both typing information and "freshness" (or better, "distinctness") assumptions about name-types variables. Notice that here a is a *variable* symbol, on a par with x; we use the letters a, b instead of x, y for denoting that these variables are subject to freshness constraints.

We write $\Sigma \vdash t : \tau$ to indicate that t is a well-formed term (of type τ). Type-checking rules are as usual; freshness information is irrelevant for typing terms. In particular, the rule for constants is the following

$$\frac{(c{:}(\tau_1, \ldots, \tau_n) \to \delta) \in C}{\Sigma \vdash c : \tau_1 \to \cdots \to \tau_n \to \delta}$$

We denote by $Tm_\Sigma = \{t \mid \Sigma \vdash t : \tau\}$ the set of well-formed terms in a context Σ.

Finally, the formulas of Nominal Logic are the following:

$$\begin{aligned}\phi, \psi ::= {} & \top \mid \bot \mid p(\vec{t}) \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \supset \psi \\ & \mid t \approx u \mid a\#t \mid \forall x{:}\tau.\phi \mid \exists x{:}\tau.\phi \mid \mathsf{N}a{:}\nu.\phi\end{aligned}$$

where p ranges over propositional symbols in P. The intuitive meaning of the proposition $a\#t$ is "a is not free in t" (more precisely, a is not in the support of t). In $\mathsf{N}a{:}\nu.\phi$, a is bound; notice that the bound symbol is a *variable* a, *not* a constant a.

Well-formedness of formulas is denoted by the judgment $\Sigma \vdash \phi\ \mathit{form}$. The rules are as usual; notice that freshness information in Σ is *not* needed for typing atomic propositions, but it is essential for typing the N quantifier:

$$\frac{\Sigma, \vec{n}_1 : \vec{\nu}_1 \vdash t_1 : \delta_1\ \ldots\ \Sigma, \vec{n}_k : \vec{\nu}_k \vdash t_k : \delta_k}{\Sigma \vdash p(\langle\vec{n}_1\rangle t_1, \ldots, \langle\vec{n}_k\rangle t_k)\ \mathit{form}} \qquad \frac{\Sigma\#a{:}\nu \vdash \phi\ \mathit{form}}{\Sigma \vdash \mathsf{N}a{:}\nu.\phi\ \mathit{form}}$$

where $p{:}(\langle\vec{\nu}_1\rangle\delta_1, \ldots, \langle\vec{\nu}_k\rangle\delta_k) \in P$.

For each context Σ, we denote by $\Sigma^\#$ the set of *freshness formulas in* Σ, defined inductively as follows:

$$\langle\rangle^\# = \emptyset \qquad (\Sigma, x{:}\tau)^\# = \Sigma^\#$$
$$(\Sigma\#a{:}\nu)^\# = \Sigma^\# \cup \{a\#t \mid t \in Tm_\Sigma\}$$

For all $\phi : \Sigma^\#$, it is $\Sigma \vdash \phi\ \mathit{form}$. $\Sigma^\#$ can be seen as the "closure" of the freshness information declared in Σ to all terms which can be defined in Σ. For instance, $(x{:}\tau\#a{:}\nu)^\# = \{a\#t \mid x{:}\tau \vdash t : \tau\}$. Notice that the set $\Sigma^\#$ is parametric in the given signature $\mathcal{S}$.

The usual weakening and substitution lemmas hold [1].

3.2 The rules

The derivation judgments have the form $\Sigma : \Gamma \Rightarrow \phi$, where Σ is a context, Γ is a set of formulas and ϕ is a formula. The judgment is well-formed if and only if for all formula $\psi \in \Gamma, \phi$: $\Gamma \vdash \psi\ \mathit{form}$.

Similarly to $\mathrm{NL}^\Rightarrow$ [1], the proof system is composed by a(ny) standard set of rules for intuitionistic first order logic with equality (which we omit here), extended with

- a rule for introducing (instances of the) axioms about freshness, equivariance and abstraction;
- a rule for introducing freshness formulas ($\Sigma\#$);
- a structural rule for freshness (*Fresh*);
- two rules $\mathsf{N}\mathcal{I}$, $\mathsf{N}\mathcal{E}$ for the N quantifier.

Axioms and rules are given in Figure 1 and 2 respectively. These axioms and rules state the fundamental properties of swapping, freshness and equivariance; for instance, E_3 states that atomic propositions of the given nominal signature must be equivariant (i.e., they are not sensible to swapping of names in terms). Notice that the proof system is parametric in the nominal signature, since $\Sigma\#$ introduces formulas from the set $\Sigma^\#$.

It is easy to see that NINL is equivalent to the intuitionistic version of $\mathrm{NL}^\Rightarrow$, in virtue of the cut elimination for the latter [1]. Therefore, NINL is a conservative extension of the original Pitts' system for Nominal Logic [9]. One difference between $\mathrm{NL}^\Rightarrow$ and NINL is that in the latter, A_2 and A_3 are axioms instead of rules.

(S_1) $(a\ a) \cdot x \approx x$
(S_2) $(a\ b) \cdot (a\ b) \cdot x \approx x$
(S_3) $(a\ b) \cdot a \approx b$
(E_1) $(a\ b) \cdot c \approx c$
(E_2) $(a\ b) \cdot (t\ u) \approx ((a\ b) \cdot t)((a\ b) \cdot u)$
(E_3) $p(\vec{x}) \supset p((a\ b) \cdot \vec{x})$
(E_4) $(a\ b) \cdot \lambda x{:}\tau.t \approx \lambda x{:}\tau.(a\ b) \cdot t[((a\ b) \cdot x)/x]$
(F_1) $a\#x \wedge b\#x \supset (a\ b) \cdot x \approx x$
(F_2) $a\#b \quad (a{:}\nu, b{:}\nu', \nu \neq \nu')$
(F_3) $a\#a \supset \bot$
(F_4) $a\#b \vee a \approx b$
(A_1) $a\#y \wedge x \approx (a\ b) \cdot y \supset \langle a\rangle x \approx \langle b\rangle y$
(A_2) $\langle a\rangle x \approx \langle b\rangle y \supset (a \approx b \wedge x \approx y) \vee (a\#y \wedge x \approx (a\ b) \cdot y)$
(A_3) $\forall y : \langle\nu\rangle\tau \exists a : \nu \exists x : \tau.y \approx \langle a\rangle x$

Figure 1. Equational and freshness axioms of NINL.

$$\frac{}{\Sigma : \Gamma \Rightarrow \phi}\ Ax \quad \phi \text{ instance of some axiom}$$

$$\frac{\Sigma\#a{:}\nu : \Gamma \Rightarrow \phi}{\Sigma : \Gamma \Rightarrow \phi}\ Fresh \qquad \frac{\phi \in \Sigma^{\#}}{\Sigma : \Gamma \Rightarrow \phi}\ \Sigma\#$$

$$\frac{\Sigma\#a{:}\nu : \Gamma \Rightarrow \phi}{\Sigma : \Gamma \Rightarrow \mathsf{N}a.\phi}\ \mathsf{N}\mathcal{I} \qquad \frac{\Sigma : \Gamma \Rightarrow \mathsf{N}a.\phi \quad \Sigma\#a{:}\nu : \Gamma, \phi \Rightarrow \psi}{\Sigma : \Gamma \Rightarrow \psi}\ \mathsf{N}\mathcal{E}$$

Figure 2. Rules of $\mathsf{NINL}(\mathcal{S})$.

4. Nominal signatures in CIC/ToC

In this section, we present the encoding of a nominal signature as a signature of the CIC type theory. The encoding is "shallow", in the sense that the logic for reasoning about terms of the nominal signature will be the same logic of CIC.

Following [4], the encoding is in three steps:

1. the encoding of the syntax of terms and formulas;
2. the definition of the "non-occurrence predicates" `notin` which is defined in a syntax-driven way;
3. the encoding of the axioms of the Theory of Contexts for the given signature; these axioms use the `notin` predicates previously defined.

By adding the resulting signature to the CIC type theory we obtain a proof system tailored for reasoning about the given nominal signature $\mathcal{S}$. We call this system $\mathsf{CIC/ToC}(\mathcal{S})$, and denote its typing judgment as

$$x : t_1, \ldots, x : t_n \vdash_{\mathsf{ToC}(\mathcal{S})} d : t$$

In the rest of this section we review the three steps above; for the sake of definiteness, we use the Coq syntax for CIC [6].

4.1 Syntax of terms and formulas

The first step for encoding a nominal signature $\mathcal{S} = (N, D, C, P)$ in CIC is to introduce one parametric type for each of the $\nu_i \in N$. Thus we have the following declarations:

```
Parameter Name_1: Set.
Parameter Name_2: Set.
...
Parameter Name_n: Set.
```

Since we will represent binders following a "weak"-HOAS style, none of the `Name_i` can be encoded by means of a (co)inductive type in order to avoid "exotic" terms and inconsistencies with the axioms of ToC. Names are then represented by CIC metavariables of these types.

Then, for each $\delta_i \in D$ we introduce a (mutually) inductive type `delta_i` of sort `Set`, in order to take advantage of the inductive features offered by CIC (e.g., automatically generated induction principles, case analysis etc.)[2].

```
Mutual Inductive delta_1: Set :=  ...
...
  with Inductive delta_j: Set :=  ...
```

Each constructor $c_i{:}\sigma \to \delta \in C$ allows to build terms of a data type δ from other terms as specified by σ. Thus, each constructor $c_i{:}\sigma \to \delta$ is represented by a constructor `c_i:`$Curry(\sigma)$`->delta` in the declaration of the type `delta`, where the "curryfication" function is as usual. For instance, the representation of the nominal signature for the untyped λ-calculus given at the end of Section 2 can be expressed as follows:

[2] We can consider also *coinductive* types, like e.g. streams, although they are not covered by the current definition of nominal signatures.

```
Parameter Var: Set.
Inductive Term: Set :=
    var: Var -> Term
  | lam: (Var -> Term) -> Term
  | app: Term -> Term -> Term.
```

Notice that, in order to rule out exotic terms, binders (as `lam` above) are rendered by constants whose arguments are functions over the open types of names, according to the weak HOAS paradigm.

Atomic predicates are represented similarly, i.e., by declaring (possibly mutual) (co)inductive predicates `p_i:`$Curry(\sigma_i)$`->Prop` corresponding to the propositional symbols $p_i{:}\sigma_i \in P$. Since we adopt a "shallow" encoding, we use the CIC sort `Prop` for representing propositions and predicates.

4.2 Non-occurrence predicates

In the previous subsection we have described how to represent a nominal signature in CIC, using the "weak" higher-order abstract syntax paradigm. In order to adequately reason about name properties, it is important to have the possibility to express statements about the (free) occurrence/non-occurrence of names into terms or contexts. This problem is addressed by defining a binary predicate `notin` such that `(notin x t)` holds iff the name `x` does not occur free into the term `t`. Clearly, we have to introduce a distinct `notin` predicate for each data type symbol δ_i. Moreover, the introduction clauses for the `notin` predicate at hand must be specified in a syntax driven way, where each δ_i-constructor generates one or more rules, according to the following definition:

Definition 3 *Let $\mathcal{S} = (N, D, C, P)$ be a nominal signature, and $\nu \in N$, $\delta \in D$ a name type and data type respectively represented in $\mathsf{CIC/ToC}(\mathcal{S})$ by* `Name` *and* `delta`*. The* `notin_delta` *predicate is defined as follows:*

```
Inductive notin_delta (x:Name): delta -> Prop :=
...
| notin_c_i: forall t1:tau1, ..., forall tk:tauk,
              N_1 -> ... -> N_k ->
              (notin_tau x (c_i t1 ... tk))
...
```

where for each constructor `c_i:tau1->...->tauk->delta` *of* `delta`*, there is a clause* `notin_c_i` *defined as follows:*

- *if* `tau_i` $=$ `Name`*, then* `N_i`$\triangleq$`x<>ti`*;*
- *if* `tau_i = Name_i1 -> ... -> Name_ih -> delta_i`*, then* `N_i` *is as follows:*

```
forall y1:Name_i1,...,forall yh:Name_ih,
       x<>y1 -> ... -> x<>yh ->
       (notin_tau x (ti y1 ... yh))
```

For instance, in our running example about the untyped λ-calculus, we have:

```
Inductive notin_Term (x:Var): Term -> Prop :=
  notin_var: forall y:Var, x<>y ->
              (notin_Term x (var y))
| notin_lam: forall t: Var -> Term,
              (forall y:Var, x<>y ->
                  (notin_Term x (t y)))
              -> (notin_Term x (lam t))
| notin_app: forall t1 t2: Term,
              (notin_Term x t1) ->
              (notin_Term x t2) ->
              (notin_Term x (app t1 t2)).
```

4.3 The Theory of Contexts

The aim of the Theory of Contexts [4] is to reflect on a formal level the structural properties of data structures with binders which are taken for granted when reasoning "on paper". In particular, the theory consists of a small set of "natural" properties about names and *(syntactic) contexts*, that is terms with "holes" which can be filled with variables/names to become plain terms; in particular, these contexts are represented as functions over sets of variables/names.

The first property we assume is that there is an infinite supply of names of each name type; that is, given any term there exists always a name fresh with respect to it (i.e. a name not occurring in the given term):

```
Axiom fresh_i: forall t:tau, exists a:Name_i,
             (notin_tau a t).
```

The intuition behind this axiom is that terms are finite objects; hence, a single term cannot contain all the possible names. Moreover, since we are interested in meta-reasoning over nominal languages, we want the equality over names to be decidable. This property can be stated formally as follows:

```
Axiom Name_i_dec_i: forall a b:Name_i, a=b \/ a<>b.
```

In a classical context, `Name_i_dec` is obviously an instance of the law of excluded middle; in this case, we do not need to assume it explicitly. On the other hand, in an intuitionistic setting it represents the minimum classical flavour needed in order to allow a meta-theoretic reasoning about encodings of nominal languages.

In the case of object systems with several name types, we have also to assume that different name types are disjoint. For each `Name_i`, `Name_j` previously declared, with $i < j$, we assume

```
Axiom Name_ij : forall a:Name_i, forall b:Name_j,
                (notin a b).
```

Usually, the implementations of the most widely used Logical Frameworks do not provide any support for reasoning about term contexts, that is functional terms. In fact, neither induction nor recursion principles over higher-order terms are available. Hence, it is practically impossible to carry out formal proofs by reasoning over the structure of contexts. In order to overcome this problem, the Theory of Contexts assumes the the axioms of β-*expansion* and *extensionality over names:*

```
Axiom tau_exp: forall t:tau, forall x:Name,
               exists t':Name->tau,
               (notin_tau_ho x t') /\ t=(t' x).
Axiom tau_ext: forall f g:Name->tau,forall x:Name,
               (notin_tau_ho x f) ->
               (notin_tau_ho x g) ->
               (f x)=(g x) -> f=g.
```

where `notin_tau_ho` is the following definition:

```
Definition notin_tau_ho:=
  fun x:Name => fun f:Name->tau =>
    (forall y: Name, x<>y -> (notin_tau x (f y))).
```

Intuitively, the axiom of β-expansion `tau_exp` allows to "split" a term `t` into a context `t'` for a given name `x`, whereas the axiom of extensionality `tau_ext` allows to state the equality of two contexts when their applications to a fresh name yield the same term. Thus, we have a rather simple machinery allowing us to introduce and to reason about syntactical properties of contexts.

It is worth noticing that the axioms of β-expansion and extensionality can be generalized to contexts with an arbitrary number of "holes".

5. Translating NINL specifications to CIC/ToC

Let us denote by $\mathcal{S} = (N, D, C, P)$ a given nominal signature. In this section we describe a translation of types, terms, formulas and judgments of NINL over $\mathcal{S}$ into types, terms and propositions of CIC/ToC with the signature of $\mathcal{S}$, respectively.

5.1 Translation of types and signatures

Each type τ of NINL is easily translated into a type $[\![\tau]\!]$ of sort `Set` of CIC, as follows:

Translation of types

$$
\begin{aligned}
[\![\delta_i]\!] &= \texttt{delta\_}i \quad (\delta \in D)\\
[\![\nu_i]\!] &= \texttt{Name\_}i \quad (\nu_i \in N)\\
[\![\tau \to \tau']\!] &= [\![\tau]\!]\texttt{->}[\![\tau']\!]\\
[\![\langle\nu\rangle\tau]\!] &= [\![\nu]\!]\texttt{->}[\![\tau]\!]
\end{aligned}
$$

where `delta` : `Set` is the `Inductively`-defined type in the signature of $\mathcal{S}$ corresponding to $\delta \in D$ (Section 4).

Proposition 1 *For each type τ of NINL($\mathcal{S}$): $\vdash_{ToC(\mathcal{S})} [\![\tau]\!]$: `Set`.*

A typing signature Σ of NINL is translated to a sequence of variable declarations (i.e., a signature in CIC), equipped with suitable assumptions for representing freshness information:

Translation of signatures

$$
\begin{aligned}
[\![\langle\rangle]\!] &= \emptyset \qquad \text{(no declarations)}\\
[\![\Sigma, x:\tau]\!] &= [\![\Sigma]\!]\ \texttt{Variable x:}[\![\tau]\!]\texttt{.}\\
[\![\Sigma \# a:\nu]\!] &= [\![\Sigma]\!]\ \texttt{Variable a:}[\![\nu]\!]\texttt{.}\\
&\quad\ \texttt{Hypothesis fresh\_a:}\\
&\quad\ \texttt{(notin a x1)/\.../\(notin a xn).}\\
&\quad\ (\text{where } \mathrm{dom}(\Sigma) = \{x_1, \ldots, x_n\})
\end{aligned}
$$

5.2 Translation of terms

Although most of the translation of terms is easy, it is here where the difference between the first-order approach of NINL and the second-order flavour of weak HOAS becomes evident. Let us discuss first the case of name abstraction. In nominal logics, names are not bound by name abstractions: *abs* is a plain constructor, thus a is free in $\langle a\rangle b$ (actually, a can be any term of type ν, not only a variable). On the other hand, in weak HOAS name abstraction corresponds to functional abstraction (as reflected by the resulting type `Name->tau`). Thus, a term $\langle a\rangle t$ must be mapped to some *functional* term `u` of CIC, such that `(u a)` corresponds exactly to t and moreover `a` does not appear free in `u`. The problem is, how to define such u if t is not a closed term, e.g. if it is a variable? To circumvent this problem, we do not define this term `u` directly from t in the translation; instead, we *assume* that it is provided (together with its properties) by the (translation of the) formula surrounding the term. For instance, the translation of a formula $p((a\ b)\cdot t, \langle c\rangle t')$ will be a proposition of the form

```
forall y:Name->Name->tau, (y a b)=t -> ... ->
forall z:Name->tau', (z c)=t' -> ... ->
(p (y b a) z)
```

(As we will see, the translation of formulas provides also suitable "freshness" information about these functional variables, but this is not relevant for translation of terms).

Therefore, the translation of a term is twofold: on one side, it translates a NL term into a CIC expression; on the other, it produces also a set of fresh variables and equational conditions, needed for this expression to make sense.

Let us consider first the second part. Given a term t, we can calculate the set of fresh variables and suitable equations by looking at all name swapping and name abstractions occurring in t; if these subterms occur within a λ-abstraction, then the corresponding equations must be abstracted accordingly. More formally, for each term t of NINL($\mathcal{S}$), we denote by $\mathcal{E}(t)$ a set of equational proposition of the following forms:

$$\lambda x_1{:}\tau_1 \dots \lambda x_n{:}\tau_n.(y\ x_1\ \dots\ x_m\ a) = u$$
$$\text{or}\quad \lambda x_1{:}\tau_1 \dots \lambda x_n{:}\tau_n.(y\ x_1\ \dots\ x_m\ a\ b) = u$$

where y is a variable and $n \geq m \geq 0$. $\mathcal{E}(t)$ is defined by induction on the syntax of t, as follows:

Calculation of equational propositions

$$\begin{aligned}
\mathcal{E}(x) =& \mathcal{E}(\mathtt{a}) = \mathcal{E}(c) = \emptyset \\
\mathcal{E}(t\ u) =& \mathcal{E}(t) \cup \mathcal{E}(u) \\
\mathcal{E}(\langle a\rangle t) =& \mathcal{E}(a) \cup \mathcal{E}(t) \cup \{(y\ a) = t\} (y \text{ fresh}) \\
\mathcal{E}((a\ b)\cdot t) =& \mathcal{E}(a) \cup \mathcal{E}(b) \cup \mathcal{E}(t) \cup \{(y\ a\ b) = t\} \qquad (y \text{ fresh}) \\
\mathcal{E}(\lambda x{:}\tau.t) =& \{\lambda x\vec{z}{:}\tau\vec{\sigma}.(y\ x\ \vec{z}\ a) = \lambda x{:}\tau.u \mid \\
& \qquad (\lambda\vec{z}{:}\vec{\sigma}.(y\ \vec{z}\ a) = u) \in \mathcal{E}(t)\} \cup \\
& \{\lambda x\vec{z}{:}\tau\vec{\sigma}.(y\ x\ \vec{z}\ a\ b) = \lambda x{:}\tau.u \mid \\
& \qquad (\lambda\vec{z}{:}\vec{\sigma}.(y\ \vec{z}\ a\ b) = u) \in \mathcal{E}(t)\}
\end{aligned}$$

(We can calculate from $\mathcal{E}(t)$ the set of fresh variables to allocate, by collecting the head variables on the left hand side of equations.)

Then, we can define the function mapping well-formed terms of NINL($\mathcal{S}$) into terms of CIC:

$$[\![\cdot]\!]^{\Phi}_{\Sigma} : Tm_{\Sigma} \to \{\mathtt{t} \mid \mathtt{t} \text{ well-formed term of CIC/ToC}(\mathcal{S})\}$$

where Σ is a typing signature of NINL, and Φ is a set of equational propositions rich enough to make the translation well-defined (like, in particular, those calculated by $\mathcal{E}$). The translation is defined by induction on the typing derivations $\Sigma \vdash t : \tau$, which in turn is by induction on the syntax of t. Let us consider the interesting cases.

- A name abstraction $\langle a\rangle t$ (of type $\langle\nu\rangle\tau$) is mapped to a term u of type `Name->tau`, representing the term t where a has been "taken away" (leaving an hole in it). Hence, u must appear in the right hand side of an equation $(u\ a) = t$ in Φ. Thus

$$[\![\langle a\rangle t]\!]^{\Phi}_{\Sigma} = (\mathtt{y\ x1} \dots \mathtt{xm}) \qquad \text{for } ((y\ x_1\ \dots x_m\ a) = t) \in \Phi$$

- The swapping $(a\ b)\cdot t$ can be seen as the result of "filling" the two "holes" of a term $t' : \nu \to \nu \to \tau$, such that $(t'\ a\ b) = t$, with b and a respectively. Thus

$$[\![(a\ b)\cdot t]\!]^{\Phi}_{\Sigma} = (\mathtt{y\ x1} \dots \mathtt{xm}\ [\![b]\!]^{\Phi}_{\Sigma}\ [\![a]\!]^{\Phi}_{\Sigma}) \quad \text{for } ((y\ x_1 \dots x_m\ a\ b) = t) \in \Phi$$

- The translation of a functional abstraction $\lambda x{:}\tau.t$ adds a new, local variable x to the signature used for translating t. This variable can be used for instantiating equations in Φ about abstracted terms. Thus

$$[\![\lambda x{:}\tau.t]\!]^{\Phi}_{\Sigma} = \mathtt{fun\ x:}[\![\tau]\!]\ \mathtt{=>}\ [\![t]\!]^{\Phi@(x:\tau)}_{\Sigma,x:\tau}$$

where $\Phi@(x{:}\tau)$ is an "instantiation" of propositions in Φ between abstracted terms (of the matching type):[3]

$$\Phi@(x{:}\tau) \triangleq \Phi \cup \{(u\ x) = (t\ x) \mid (u = t) \in \Phi, u \text{ of type } \tau \to \tau' \text{ for some } \tau'\}$$

The remaining cases (variable, constants and application) are trivial. Summarizing, we have:

Translation of terms

$$\begin{aligned}
[\![x]\!]^{\Phi}_{\Sigma} &= \mathtt{x} \\
[\![c]\!]^{\Phi}_{\Sigma} &= \mathtt{c} \qquad (\text{for } c \in C) \\
[\![\mathtt{a}]\!]^{\Phi}_{\Sigma} &= \mathtt{a} \\
[\![tu]\!]^{\Phi}_{\Sigma} &= (\mathtt{[\![}t\mathtt{]\!]}^{\Phi}_{\Sigma}\ \mathtt{[\![}u\mathtt{]\!]}^{\Phi}_{\Sigma}) \\
[\![\lambda x{:}\tau.t]\!]^{\Phi}_{\Sigma} &= \mathtt{fun\ x:[\![}\tau\mathtt{]\!]\ =>\ [\![}t\mathtt{]\!]}^{\Phi@(x:\tau)}_{\Sigma,x:\tau} \\
[\![\langle a\rangle t]\!]^{\Phi}_{\Sigma} &= [\![u]\!]^{\Phi}_{\Sigma} \qquad \text{where } ((u\ a) = t) \in \Phi \\
[\![(a\ b)\cdot t]\!]^{\Phi}_{\Sigma} &= (\mathtt{[\![}u\mathtt{]\!]}^{\Phi}_{\Sigma}\ \mathtt{[\![}b\mathtt{]\!]}^{\Phi}_{\Sigma}\ \mathtt{[\![}a\mathtt{]\!]}^{\Phi}_{\Sigma}) \qquad \text{where } ((u\ a\ b) = t) \in \Phi
\end{aligned}$$

Notice that the free variables of $[\![t]\!]^{\Phi}_{\Sigma}$ may come either from Σ or from Φ. Therefore, in order to state the adequacy property of this translation, we need to define a map $[\![\cdot]\!]$ from sets of equations Φ to signatures of CIC containing both the declaration of variables, and the freshness and equational properties:

- $[\![\emptyset]\!]_{\Sigma} = \emptyset$ (the empty signature);
- for $\Sigma \vdash u : \vec{\sigma} \to \tau$:

$$\begin{aligned}
&[\![\Phi, \lambda\vec{x}{:}\vec{\sigma}.(y\ \vec{x}\ a){=}u]\!]_{\Sigma} = \\
&\quad [\![\Phi]\!]_{\Sigma} \\
&\quad \mathtt{Variable\ y:[\![}\vec{\sigma} \to \nu \to \tau\mathtt{]\!]}. \\
&\quad \mathtt{Hypothesis\ y_eq:[\![}\lambda\vec{x}{:}\vec{\sigma}.(y\ \vec{x}\ a)\mathtt{]\!]}^{\Phi}_{\Sigma}\mathtt{=[\![}u\mathtt{]\!]}^{\Phi}_{\Sigma}. \\
&\quad \mathtt{Hypothesis\ y_fresh:(notin\ [\![}a\mathtt{]\!]}^{\Phi}_{\Sigma}\ \mathtt{y)}.
\end{aligned}$$

- for $\Sigma \vdash u : \vec{\sigma} \to \tau$:

$$\begin{aligned}
&[\![\Phi, \lambda\vec{x}{:}\vec{\sigma}.(y\ \vec{x}\ a\ b){=}u]\!]_{\Sigma} = \\
&\quad [\![\Phi]\!]_{\Sigma} \\
&\quad \mathtt{Variable\ y:[\![}\vec{\sigma} \to \nu \to \nu \to \tau\mathtt{]\!]} \\
&\quad \mathtt{Hypothesis\ y_eq:[\![}\lambda\vec{x}{:}\vec{\sigma}.(y\ \vec{x}\ a\ b)\mathtt{]\!]}^{\Phi}_{\Sigma}\mathtt{=[\![}u\mathtt{]\!]}^{\Phi}_{\Sigma}. \\
&\quad \mathtt{Hypothesis\ y_fresh1:(notin\ [\![}a\mathtt{]\!]}^{\Phi}_{\Sigma}\ \mathtt{y)}. \\
&\quad \mathtt{Hypothesis\ y_fresh2:(notin\ [\![}b\mathtt{]\!]}^{\Phi}_{\Sigma}\ \mathtt{y)}.
\end{aligned}$$

Freshness and equational properties will be needed in the next sections; they are not needed for the following result:

Proposition 2 *For all terms $t \in Tm_{\Sigma}$ and types τ in NINL($\mathcal{S}$): $\Sigma \vdash t : \tau$ iff $[\![\Sigma]\!], [\![\mathcal{E}(t)]\!]_{\Sigma} \vdash_{\mathcal{T}oC(\mathcal{S})} [\![t]\!]^{\mathcal{E}(t)}_{\Sigma} : [\![\tau]\!]$.*

5.3 Translation of formulas

Formulas of NINL are translated to terms of CIC inhabiting the sort `Prop`. The mapping is straightforward except for the И quantifier and the base case.

[3] The reader aware of Kripke models and/or presheaf models of variables, will recognize here a similarity with Kripke semantics for S4. Indeed, the Σ denotes the current world, and Φ the "level of knowledge" at the current world. Equivalence about abstracted terms are equivalences about the instantiated terms, in all reachable worlds. Entering a *variable* abstraction is like moving to a new world, where the knowledge may change (consistently with that of the world we come from), hence the instantiation.

Translation of formulas

$$
\begin{aligned}
[\![\top]\!]_\Sigma &= \texttt{True}\\
[\![\bot]\!]_\Sigma &= \texttt{False}\\
[\![\phi \wedge \psi]\!]_\Sigma &= [\![\phi]\!]_\Sigma \ \texttt{/\textbackslash}\ [\![\psi]\!]_\Sigma\\
[\![\phi \vee \psi]\!]_\Sigma &= [\![\phi]\!]_\Sigma \ \texttt{\textbackslash/}\ [\![\psi]\!]_\Sigma\\
[\![\phi \supset \psi]\!]_\Sigma &= [\![\phi]\!]_\Sigma \ \texttt{->}\ [\![\psi]\!]_\Sigma\\
[\![\forall x{:}\tau.\phi]\!]_\Sigma &= \texttt{forall x:}[\![\tau]\!]\texttt{,}\ [\![\phi]\!]_{\Sigma,x:\tau}\\
[\![\exists x{:}\tau.\phi]\!]_\Sigma &= \texttt{exists x:}[\![\tau]\!]\texttt{,}\ [\![\phi]\!]_{\Sigma,x:\tau}\\
[\![\mathsf{И}a{:}\nu.\phi]\!]_\Sigma &= \texttt{forall a:Name,}\\
&\quad \texttt{(notin a x1)->...->(notin a xn)->}[\![\phi]\!]_{\Sigma,a:\nu}\\
&\quad \text{where } \mathrm{dom}(\Sigma) = \{x_1,\ldots,x_n\}\\
[\![p(\vec{t})]\!]_\Sigma &= \text{(see below)}
\end{aligned}
$$

The translation of $\mathsf{И}a{:}\nu.\phi$ is similar to that of fresh name variables in signatures, taking advantage of the "for all" flavor of И: the fresh name is translated to a universally-quantified variable, equipped with suitable freshness assumptions with respect to the global context. Notice that in this case the local variable a is added to the signature using a comma and not a $\#$, because this freshness information is already encoded by the local assumptions.

The translation of atomic propositions $p(t_1,\ldots,t_n)$ is more subtle. We have to translate the terms $t_1,\ldots,t_n$ as well, and according to the translation of Section 5.2, this requires to introduce all the term schemata (i.e., term abstracted over `Name`), together with the suitable hypothesis, needed for translating every name swapping and name abstractions appearing in terms. For instance, the translation of $p((a\ b)\cdot t, \langle c\rangle t')$ will be

```
forall y:Name->Name->tau,
  (y a b)=t -> (notin a y) -> (notin b y) ->
forall z:Name->tau',
  (z c)=t' -> (notin c z) ->
(p (y b a) z)
```

In general, it is possible to determine the fresh variables and local hypothesis required for translating an atomic proposition, just by applying the function $\mathcal{E}$ of Section 5.2 to the inner terms using. The complete set of equations required for translating the proposition $p(\vec{t})$ is defined as $\Phi(\vec{t}) = \bigcup_i \mathcal{E}(t_i)$. Given such a set Φ, we define the

Translation of atomic propositions

$$
\begin{aligned}
[\![p(\vec{t})]\!]^\Phi_\Sigma &= \texttt{(p}\ [\![t_1]\!]^\Phi_\Sigma \ldots [\![t_n]\!]^\Phi_\Sigma\texttt{)} \quad \text{where } p \in P\\
[\![a\#t]\!]^\Phi_\Sigma &= \texttt{(notin}\ [\![a]\!]^\Phi_\Sigma\ [\![t]\!]^\Phi_\Sigma\texttt{)}\\
[\![t_1 \approx t_2]\!]^\Phi_\Sigma &= [\![t_1]\!]^\Phi_\Sigma\ \texttt{=}\ [\![t_2]\!]^\Phi_\Sigma
\end{aligned}
$$

We have already defined the translation of sets of equational propositions into CIC signatures (Section 5.2). We have:

Proposition 3 *For all $p(\vec{t})$ and Σ of NINL($\mathcal{S}$):*
$\Sigma \vdash p(\vec{t})$ *form if and only if* $[\![\Sigma]\!], [\![\Phi]\!]_\Sigma \vdash_{\mathsf{ToC}(\mathcal{S})} [\![p(\vec{t})]\!]^\Phi_\Sigma$ `:Prop`.

Notice that the translation of sets of equational propositions introduces also the freshness and equational hypothesis. Thus:

> the traslation $[\![p(\vec{t})]\!]$ is defined as the `Prop`osition obtained by abstracting $[\![p(\vec{t})]\!]^\Phi_\Sigma$ over all the hypothesis in $[\![\Phi]\!]_\Sigma$, so that the only hypothesis on the left of $\vdash_{\mathsf{ToC}(\mathcal{S})}$ are $[\![\Sigma]\!]$.

In other words, for each equation in Φ we add in front of the proposition $[\![p(\vec{t})]\!]^\Phi_\Sigma$, a `forall` quantification together with two or three equational and fresness hypothesis.

Proposition 4 *For all $p(\vec{t})$ and Σ of NINL($\mathcal{S}$): $\Sigma \vdash p(\vec{t})$ form if and only if* $[\![\Sigma]\!] \vdash_{\mathsf{ToC}(\mathcal{S})} [\![p(\vec{t})]\!]_\Sigma$ `: Prop`.

Proposition 5 *For all ϕ of NINL($\mathcal{S}$): $\Sigma \vdash \phi$ form if and only if* $[\![\Sigma]\!] \vdash_{\mathsf{ToC}(\mathcal{S})} [\![\phi]\!]_\Sigma$ `: Prop`.

6. The translation preserves derivability

Using the translation presented in Section 5, we can now state formally the question:

> **Soundness:** given a nominal signature $\mathcal{S}$, can we prove in CIC/ToC($\mathcal{S}$) all the properties that can be proved in NINL($\mathcal{S}$)?

Let us introduce a notion of translation for sequents of NINL. In the rest of the section, we suppose to fix a given signature $\mathcal{S}$; we will occasionally omit it from NINL($\mathcal{S}$) and CIC/ToC($\mathcal{S}$).

Definition 4 *Let $\Sigma : \Gamma \Rightarrow \phi$ be a sequent of NINL($\mathcal{S}$). We say that the sequent $\Sigma : \Gamma \Rightarrow \phi$ is* derivable in CIC/ToC *if there exists a term* `d` *of CIC/ToC($\mathcal{S}$) such that* $[\![\Sigma]\!] \vdash_{\mathsf{ToC}(\mathcal{S})}$ `d` $: [\![\bigwedge \Gamma \supset \phi]\!]_\Sigma$.

If the propositions of the nominal signature are "well-behaved", i.e., they are *equivariant*, we have that the translation indeed preserves derivability.

Theorem 1 *For all Γ, ϕ in NINL, if $\Sigma : \Gamma \Rightarrow \phi$ is derivable in NINL then $\Sigma : \Gamma \Rightarrow \phi$ is derivable in CIC/ToC.*

If we think of the translation $[\![\cdot]\!]$ as giving a *semantics* to NINL within CIC/ToC, this property can be seen as a form of *soundness* of NINL: everything derivable (in NINL) is true (in CIC/ToC).

The proof of the theorem consists in showing that the translations of all axioms and rules of NINL($\mathcal{S}$) are derivable (as `Lemma`ta) in CIC/ToC($\mathcal{S}$).[4] Let us see the various cases.

6.1 Derivability of the axioms of NINL

Lemma 1 (Axioms) *Let ϕ be an (universally closed) axiom of NINL. Then, the sequent $\emptyset : \emptyset \Rightarrow \phi$ is derivable in CIC/ToC.*

Actually, all the axioms of NINL apart from A_1, A_2 and E_3 are derivable directly from the axioms of the Theory of Contexts for a generic data type `tau`, without using specific properties of the given object language. For instance, let us see how the axioms S_2 and F_1 are rendered by our translation:

```
Lemma S2: forall x: tau, forall a b: Name,
          forall y1: Name -> Name -> tau,
          (notin_tau_ho2 a y1) ->
          (notin_tau_ho2 b y1) ->
          forall y2: Name -> Name -> tau,
          (notin_tau_ho2 a y2) ->
          (notin_tau_ho2 b y2) ->
          (y2 a b)=x -> (y1 a b)=(y2 b a) ->
          (y1 b a)=x.

Lemma F1: forall a b: Name, forall x: tau,
          forall y: Name->Name->tau,
          (notin_tau a x) -> (notin_tau b x) ->
          (notin_tau_ho2 a y) ->
          (notin_tau_ho2 b y) ->
          (y a b)=x -> (y b a)=x.
```

[4] See `http://www.dimi.uniud.it/scagnett/Coq-Sources/nlintoc.v` for the Coq code of these proofs.

Axioms A_1**,** A_2 The proofs of A_1 and A_2 are similiar, and both need the structural inductions on terms. Let us consider A_1, which is translated in CIC/ToCas follows:

```
Lemma A1: forall a:Name, forall b:Name,
          forall x:tau, forall y:tau,
          forall y':Name->Name->tau,
          (notin_tau a y) /\
          (notin_tau a (lam (fun z:Name =>
                             (lam (y' z))))) /\
          (notin_tau b (lam (fun z:Name =>
                             (lam (y' z))))) /\
          x=(y' b a) /\ y=(y' a b) ->
          exists x':Name -> tau,
          (notin_tau a (lam x')) /\
          x=(x' a) /\
          exists y'':Name -> tau,
          (notin_tau b (lam y'')) /\
          y=(y'' b) /\
          (fun a:Name=>(x' a)) =
               (fun b:Name => (y'' b)).
```

The proof of this lemma relies on the following property, stating that if a name `a` does not occur free in a context `s'` instantiated with `a` itself, then the "hole" of `s'` must be "dummy", in the sense that, filling it with whatever distinct name `b` yields the same term:

```
Lemma dummy_ctxt:
 forall s:tau, forall s':Name->tau, forall a:Name,
 s=(s' a) -> (notin_tau a (s' a)) ->
 forall b:Name, ~a=b -> (s' a)=(s' b).
```

This property can be easily proved by structural induction on `s`, using the extensionality axiom to "lift" the structural information about `s` to `s'`. Thus, the proof of `dummy_ctxt` needs the induction principle provided by CIC about the datatype `tau`. Hence, the proof of (the translation of) A_1 is parametric on the particular object language we consider (For the sake of simplicity, we proved A_1 by taking `tau` as the data type `Term` of untyped λ-terms of Section 4.1).

Axiom E_3 The axiom E_3 states the *equivariance* of atomic propositions. Naively, one could try to prove E_3 by proving a `Lemma` like the following:

```
Lemma Equivariance: forall p:tau->Prop,
    forall t:Name->tau, forall a b:Name,
    (notin a t) -> (notin b t) ->
    (p (t a)) -> (p (t b)).
```

However, it is clear that this `Lemma` does not hold: in general, it is possible to define in CIC a non-equivariant predicate `p` over `tau`. In fact, we do not need to prove `Equivariance` for a generic predicate, but only for those defined by the given nominal signature. For these predicates, we can check `Equivariance` by induction on the derivation rules on a case-by-case basis. An example of this kind of proofs is in [5], where equivariance is proved for the transition relations and bisimilarity of π-calculus.

The definition of nominal signatures adopted in this paper (Definition 2), does not considers also the issue of how atomic propositions are defined; thus, we cannot give a general proof of E_3. We defer this result to the full version of this work; for the moment, we assume that each atomic predicate is "well-behaved", that is, it is defined by an inductive definition in CIC/ToC, for which `Equivariance` (and hence E_3) is derivable. Actually, if some p of a nominal signature $\mathcal{S}$ were not equivariant, then it would contradict axiom E3 and therefore NINL($\mathcal{S}$) would be inconsistent.

6.2 Derivability of the rules of NINL

Lemma 2 (Fresh) *If* $\Sigma\#a:\nu : \Gamma \Rightarrow \phi$ *is derivable in CIC/ToC, then* $\Sigma : \Gamma \Rightarrow \phi$ *is derivable in CIC/ToC.*

PROOF. An application of the `fresh` axiom of the ToC. □

Lemma 3 ($\Sigma\#$) *Let* Σ *be a signature of NINL; then, for all* $t \in Tm_\Sigma$*, the sequent* $\Sigma\#a{:}\nu : \Rightarrow a\#t$ *is derivable in CIC/ToC.*

PROOF. In principle, since $\Sigma^\#$ can be infinite, we should prove an infinite set of `Lemma` in Coq. Actually, it is sufficient to go by induction, reflecting the definition of $\Sigma^\#$. In other words, we have to prove one `Lemma` for each base case and for each inductive step of term typing.

Let us fix the encoding of $\Sigma\#a{:}\nu$:

```
Variable x1:tau1. ... Variable xn:taun.
Variable a:Name.
Hypothesis fresh_a:(notin a x1)/\.../\(notin a xn).
```

The base cases involve variables and constants.

```
Lemma Sigma_hash_x1 : (notin a x1).
...
Lemma Sigma_hash_xn : (notin a xn).
```

which are trivial; then

```
Lemma Sigma_hash_c1 : (notin a c1).
...
Lemma Sigma_hash_ck : (notin a ck).
```

which follows from the definition of `notin`.

Then, the inductive steps are applications and λ-abstractions:

```
Lemma Sigma_hash_app:
  forall t1:tau->sigma, forall t2:tau,
  (notin a t1) -> (notin a t2) -> (notin a (t1 t2)).
```

for each base type τ. This `Lemma` can be proved by inverting the hypothesis `notin a t1`. For λ-abstraction:

```
Lemma Sigma_hash_abs:
  forall t:tau->sigma,
  (forall x:tau.(notin a x) -> (notin a (t x)))
  -> (notin a t).
```

which follows again from the definition of `notin`. □

Lemma 4 ($\forall\mathcal{I}$) *If* $\Sigma\#a:\nu : \Gamma \Rightarrow \phi$ *is derivable in CIC/ToC, then* $\Sigma : \Gamma \Rightarrow \forall a.\phi$ *is derivable in CIC/ToC.*

This lemma is a consequence of the following lemma:

```
Variable x1:tau1. ... Variable xn:taun.
Variable a:Name.
Hypothesis fresh_a:(notin a x1)/\.../\(notin a xn).
Lemma new_intro : forall phi:Name->Prop,
  (phi a) ->
  (forall b:Name, (notin a x1)->...->(notin a xn)
              -> (phi b)).
```

Essentially, this lemma says that `phi` is equivariant. As discussed in the case of the axiom E_3 above, this does not hold in general: there exist propositions in CIC which are not equivariant. However, we have assumed that equivariance can be proved for the atomic propositions defined by the given nominal signature. Under this assumption, `new_intro` is derivable in CIC/ToC.

Lemma 5 ($\forall\mathcal{E}$) *If* $\Sigma : \Gamma \Rightarrow \forall a.\phi$ *and* $\Sigma\#a:\nu : \Gamma, \phi \Rightarrow \psi$ *are derivable in CIC/ToC, then* $\Sigma : \Gamma \Rightarrow \psi$ *is derivable in CIC/ToC.*

PROOF. The thesis is a consequence of the following lemma:

```
Variable x1:tau1. ... Variable xn:taun.
Variable a:Name.
Hypothesis fresh_a:(notin a x1)/\.../\(notin a xn).
Lemma new_elim : forall P:Name->Prop, Q:Prop,
  (forall a:Name, (notin a x1)->...->(notin a xn)
              -> (P a))
  ->
  (forall a:Name, (notin a x1)->...->(notin a xn)
              -> (P a) -> Q)
  -> Q.
```

which is trivial. □

6.3 Incompleteness of NINL with respect to CIC/ToC

Once proved that the translation preserves the derivability, i.e., that NINL is correct with respect to CIC/ToC, it is natural to ask the converse:

> **Completeness:** given a nominal signature $\mathcal{S}$, if a sequent $\Sigma : \Gamma \Rightarrow \phi$ of NINL($\mathcal{S}$) is derivable in CIC/ToC($\mathcal{S}$), is it derivable in NINL($\mathcal{S}$) as well?

The answer is negative, i.e., there exist formulas of NINL which are not derivable in NINL, but whose translation is derivable in CIC/ToC. More formally:

Proposition 6 *There exists a signature $\mathcal{S}$, a closed formula ϕ of NINL($\mathcal{S}$) and a proof term* `d` *such that $\emptyset \vdash_{ToC(\mathcal{S})}$* `d` $: [\![\phi]\!]^{\emptyset}_{\emptyset}$ *and $: \Rightarrow \phi$ is not derivable in NINL.*

The counterexample is very simple, and relies on the fact that NINL is a first-order logic, while CIC is higher order—we do not even need to consider the axioms and rules about atoms and contexts. Just take a first-order signature of natural numbers, i.e. $\mathcal{S} = (\emptyset, \{nat\}, \{0 : nat, S : nat \to nat\}, \emptyset)$, and let ϕ be the formula $\phi \triangleq (0 \approx S(0)) \supset \bot$. It is clear that ϕ cannot be proved in NINL($\mathcal{S}$), because NINL is a first order logic without induction principles. On the other hand, the translation of ϕ is trivially derivable in CIC, thanks to the inductive principle automatically provided by CIC for the datatype `nat`:

```
Lemma ZeroIsNotOne: 0<>(S 0).
Proof. auto. Qed.
```

One could argue that completeness may be achieved by "weaking" the encoding in CIC/ToC. In fact, we can adopt an alternative encoding of types of the signature using "open" types *à la* LF. For instance, the type of λ-terms could be represented as follows:

```
Parameter Var: Set.
Parameter Term: Set.
Parameter var: Var -> Term.
Parameter lam: (Var -> Term) -> Term.
Parameter app: Term -> Term -> Term.
```

In this case, CIC would not provide any induction principle for `Term`. Unfortunately, we need these induction principles for proving some axioms (e.g., A_1, A_2), hence if we adopt open types instead of inductive types we are not able to prove Theorem 1.

7. Conclusions

In this paper, we have presented a work in progress about the comparison of Nominal Logics and the Theory of Contexts; in particular, we have focused on the logical expressivity of the two logics. For a class of nominal signatures endowed with equivariant propositions, we have presented a translation of terms and formulas of specifications in NL, to terms and types (of sort `Prop`) of the Calculus of Inductive Constructions, respectively. As we have shown, the translation is quite tricky due to the different nature of the two logics (first-order vs. higher order, FM-binders vs. weak HOAS, functional vs. relational approach).

We have proved that this translation preserves derivability, that is, properties derivable in NL are still derivable in CIC/ToC. On the other hand, being the former a higher-order logic with inductive types, CIC/ToC allows easily to prove more properties than Nominal Logic.

As future work, we plan to investigate some general criteria for deciding wether a predicate definition is equivariant. One possibility is to extend the notion of nominal signatures to cover also the definition of atomic predicates, and to give some syntactic restriction on the format of these definitions. For instance, in the case that atomic propositions denote operational semantics, we can consider derivation rules in standardized formats like GSOS or safe tree. Under suitable conditions about the format of these rules, it should be possible to give a general way for proving equivariance of predicates, and hence axiom E_3 within CIC/ToC.

Another possible future work is to understand if, and how, we can recover a completeness property of NINL with respect to CIC/ToC. A quite radical attempt could be to turn Nominal Logic into a full-fledged *higher-order* logic with equality, on a par with CIC/ToC; however, in this way NL would miss many of its interesting properties (such as decidability). Another possibility could be the characterization of a restricted class of proof terms in CIC/ToC (e.g., without applications of higher-order principles), which can be translated back to proofs of NINL.

Acknowledgements The authors wish to thank Pietro Di Gianantonio and the anonymous referees for their valuable suggestions.

References

[1] J. Cheney. A simpler proof theory for nominal logic. In V. Sassone, editor, *Proc. FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2005.

[2] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In G. Longo, editor, *Proc. 14th LICS*, pages 193–202, Trento, Italy, 1999. IEEE Computer Society Press.

[3] M. J. Gabbay. Fresh logic: a logic of FM. Submitted, 2003.

[4] F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on systems in higher-order abstract syntax. In *Proc. ICALP'01*, volume 2076 of *Lecture Notes in Computer Science*, pages 963–978. Springer, 2001.

[5] F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (co)inductive type theory. *Theoretical Computer Science*, 253(2):239–285, 2001.

[6] INRIA. *The Coq Proof Assistant, version 8*, 2004. Available at `http://coq.inria.fr/doc/main.html` .

[7] D. Miller and C. Palmidessi. Foundational aspects of syntax. *ACM Computing Surveys (CSUR)*, 31(3es):11, 1999.

[8] D. Miller and A. F. Tiu. A proof theory for generic judgments: An extended abstract. In *LICS 2003*, pages 118–127, Ottawa, Canada, June 2003. IEEE Computer Society.

[9] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.