An axiomatic approach to metareasoning on nominal algebras in HOAS*

Furio Honsell, Marino Miculan, and Ivan Scagnetto

Dipartimento di Matematica e Informatica,
Università di Udine, Italy
{honsell,miculan,scagnett}@dimi.uniud.it

Abstract. We present a logical framework Υ for reasoning on a very general class of languages featuring binding operators, called *nominal algebras*, presented in higher-order abstract syntax (HOAS). Υ is based on an *axiomatic* syntactic standpoint and it consists of a simple types theory *à la* Church extended with a set of axioms called the *Theory of Contexts*, recursion operators and induction principles. This framework is rather expressive and, most notably, the axioms of the Theory of Contexts allow for a smooth reasoning of schemata in HOAS. An advantage of this framework is that it requires a very low mathematical and logical overhead. Some case studies and comparison with related work are briefly discussed.
Keywords: higher-order abstract syntax, induction, logical frameworks.

Introduction

In recent years there has been growing interest in developing systems for defining and reasoning on languages featuring α-conversion. A very promising line of approach has focused on *Higher-Order Abstract Syntax* (HOAS) [7, 17, 9]. The gist of this approach is to delegate to type-theoretic metalanguages the machinery for dealing with binders. This approach however has some drawbacks. First of all, being equated to metalanguage variables, object level variables cannot be defined inductively without introducing exotic terms [2, 14]. A similar difficulty arises with contexts, which are rendered as functional terms. Reasoning by induction and definition by recursion on object level terms is therefore problematic. Finally, the major virtue of HOAS bites back, in the sense that one looses the possibility of reasoning on the properties which are delegated on the metalanguage, *e.g.* substitution and α-equivalence themselves. Various approaches have been proposed to overcome these problems based on different techniques such as modal types, functor categories, permutation models of ZF, etc. [3, 4, 8, 6, 5, 13].

The purpose of this paper is to present in broad generality yet another logical framework for reasoning on systems presented in HOAS, called Υ, based on an *axiomatic* syntactic standpoint. This system stems from the technique originally used by the authors in [10] for formally deriving in Coq [11] the metatheory of strong late bisimilarity of the π-calculus as in [16].

* Work partially supported by Italian MURST *TOSCA* project and EC-WG *TYPES*.

Υ consists of a simple types theory *à la* Church extended with a set of axioms called the *Theory of Contexts*, recursion operators and induction principles.

According to our experience, this framework is rather expressive. Higher Order Logic allows for the impredicative definition of many relations, possibly functional; the recursors and induction principles allow for the definition of many important functions and properties over contexts; and most notably the axioms in the Theory of Contexts allow for a smooth handling of schemata in HOAS.

We feel that one of the main advantages of our axiomatic approach, compared to other semantical solutions in the literature [4,6], is that it requires a very low mathematical and logical overhead. We do not need to introduce a new abstraction and concretion operators as in [6], but we can continue to model abstraction with λ-abstraction and instantiation with functional application. Therefore our approach can be easily utilized in existing interactive proof assistants, *e.g.* Coq, without needing any redesign of the system.

Of course there are some tradeoffs. One of the major theoretical problems concerning our method is the consistency of the axioms. It is here that we have to resort to more sophisticated mathematical tools, such as *functor categories*, *à la* Hofmann [8]. These are closer in spirit to [4,6], although not quite so, since our method cannot be construed as a mere axiomatization of a topos, but rather of a special tripos model. The consistency of the particular version of the system Υ tailored to the treatment of the π-calculus is proved in [1].

Another tradeoff concerns functions. Our system does not satisfy the *Axiom of Unique Choice* and hence it is functionally not very expressive. The expressive power however can be recovered using functional relations in place of functions.

In this paper we try also to outline in full generality our methodological protocol (which is the one underpinning [10]), using some well-known languages as running examples.

We feel that the present work can be useful for users of interactive proof assistants (Coq, LEGO, Isabelle [19]), as well as developers of programs/processes/ambients calculi willing to consider higher order notions in their theories, as well as implementors of proof assistants willing to extend their systems with principles for reasoning on schemata.

Synopsis. In Section 1 we introduce the notion of *nominal algebra*, together with some examples. In Section 2 we present the logical framework Υ, with the Theory of Contexts. In Section 3 we discuss first-order and higher-order recursion and induction principles in Υ. Some case studies are briefly presented in Section 4. Conclusions and comparison with related work are in Section 5.

1 Nominal algebras

In this section we present a rather general class of languages with binders, which we call *nominal algebras*. The notion of *binding signature* [4] can be viewed as a special case. Many languages we encounter in logic and computer science can be easily viewed as nominal algebras. Languages with infinitely many sorts (such

as the simply typed λ-calculus *à la* Church) or polyadic languages (such as the polyadic π-calculus) escape the proposed format. We could have easily extended it, at the expense of a more cumbersome notation, but this would have brought in side issues inessential to our purposes.

Definition 1. *A* names set υ *is an infinite enumerable set of different atomic objects, with a decidable equality. A* names base *is a finite set* $V = \{\upsilon_1, \ldots, \upsilon_k\}$ *of names sets.*

Definition 2. *Let* $V = \{\upsilon_1, \ldots, \upsilon_k\}$ *be a names base, whose elements are ranged over by* υ*. Let* $I = \{\iota_1, \ldots, \iota_m\}$ *be a set of* basic types*, ranged over by* ι*.*

A constructor arity over V, I for ι *is a type* α *of the form* $\tau_1 \times \cdots \times \tau_n \to \iota$*, where* $n \geq 0$ *and for* $i = 1 \ldots n$*, the type* τ_i *is either in* V *or it is of the form* $\tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \sigma_i$ *where* $\upsilon_{ij} \in V$ *and* $\sigma_i \in I$*. If* $m_i > 0$ *for some* i*, then* α *is said to be a* binding arity*, or to* bind $\upsilon_{i1}, \ldots, \upsilon_{im_i}$ over σ_i*.*

A constructor over V, I for ι *is a typed constant* c^α *where* α *is a constructor arity over* V, I*. If* α *is a binding arity, then* c *is said to be a* binding constructor*, or simply a* binder*.*

A nominal algebra N *is a tuple* $\langle V, I, C\rangle$ *where* V *is a set of names sets,* I *is a set of basic types, and* C *is a set of constructors over* V, I*.*

Example 1. Many languages can be viewed as nominal algebras.

- Untyped λ-calculus: $N_\lambda = \langle\{\upsilon\}, \{\Lambda\}, \{var^{\upsilon\to\Lambda}, \boldsymbol{\lambda}^{(\upsilon\to\Lambda)\to\Lambda}, app^{\Lambda\times\Lambda\to\Lambda}\}\rangle$
- First order logic (FOL): $N_{FOL} = \langle\{\upsilon\}, \{\iota, \phi\}, \{var^{\upsilon\to\iota}, 0^\iota, 1^\iota, +^{\iota\times\iota\to\iota}, =^{\iota\times\iota\to\phi}, \supset^{\phi\times\phi\to\phi}, \forall^{(\upsilon\to\phi)\to\phi}\}\rangle$
- Second Order Logic (SOL): $N_{SOL} = \langle\{\upsilon, \upsilon'\}, \{\iota, \phi\}, \{var^{\upsilon\to\iota}, var'^{\upsilon'\to\phi}, 0^\iota, S^{\iota\to\iota}, =^{\iota\times\iota\to\phi}, \supset^{\phi\times\phi\to\phi}, \forall^{(\upsilon\to\phi)\to\phi}, \Lambda^{(\upsilon'\to\phi)\to\phi}\}\rangle$
- π-calculus: $N_\pi = \langle\{\upsilon\}, \{\iota\}, \{0^\iota, |^{\iota\times\iota\to\iota}, \tau^{\iota\to\iota}, =^{\upsilon\times\upsilon\times\iota\to\iota}, \nu^{(\upsilon\to\iota)\to\iota}, in^{\upsilon\times(\upsilon\to\iota)\to\iota}, out^{\upsilon\times\upsilon\times\iota\to\iota}\}\rangle$

Definition 3. *Let* $N = \langle V, I, C\rangle$ *be a nominal algebra. The* object language generated by N*, denoted by* $\mathcal{L}(N)$ *or simply* $\mathcal{L}$*, is the set of well-typed terms definable using the names in* V *and the constructors in* C*, up-to* α*-equivalence. For* $\iota \in I$*, we denote by* $\mathcal{L}^\iota$ *the subset of* $\mathcal{L}_N$ *of terms of type* ι*.*

A stage (in $V = \{\upsilon_1, \ldots, \upsilon_k\}$) *is a tuple* $X = \langle X_1, \ldots, X_k\rangle$ *such that* $X_i \subset \upsilon_i$ *finite for* $i = 1 \ldots k$*. For* X *a stage in* V*, we denote by* $\mathcal{L}_X$ *(resp.,* $\mathcal{L}^\iota_X$*) the subset of* $\mathcal{L}$ *(resp.,* $\mathcal{L}^\iota$*) of terms with free names in* X*.*

2 The Logical Framework Υ

In this section we present the type theoretic logical system Υ for reasoning formally on nominal algebras. Many details of the underlying type theory are not strictly intrinsic. The machinery that we define in this section could have been based on any sufficiently expressive type theory, e.g., CIC. We picked Church Simple Theory of Types only for simplicity.

$$\frac{-}{\Gamma, x:\tau \vdash_\Sigma x:\tau} \quad \text{(VAR)} \qquad \frac{-}{\Gamma \vdash_\Sigma c:\tau}(c{:}\tau) \in \Sigma_c \quad \text{(CONST)}$$

$$\frac{\Gamma \vdash_\Sigma M:\tau' \to \tau \quad \Gamma \vdash_\Sigma N:\tau'}{\Gamma \vdash_\Sigma MN:\tau} \quad \text{(APP)} \qquad \frac{\Gamma \vdash_\Sigma M:o \quad \Gamma \vdash_\Sigma N:o}{\Gamma \vdash_\Sigma M \Rightarrow N:o} \quad \text{(IMP)}$$

$$\frac{\Gamma, x:\tau' \vdash_\Sigma M:\tau}{\Gamma \vdash_\Sigma \lambda x{:}\tau'.M:\tau' \to \tau} \quad \text{(ABS)} \qquad \frac{\Gamma, x:\tau \vdash_\Sigma M:o}{\Gamma \vdash_\Sigma \forall x{:}\tau.M:o} \quad \text{(FORALL)}$$

Fig. 1. Typing rules.

2.1 The Logical Framework Υ: Terms and Types

The logical framework Υ is a theory of Simple Types/Classical Higher Order Logic *à la* Church (HOL) on a given signature Σ.

A *type signature* Σ_t is a finite list of atomic type symbols $\sigma_1, \ldots, \sigma_n$.

The *simple types* over a type signature Σ_t are ranged over by τ, σ (possibly with indices or apices), and are defined by the following abstract syntax:

$$\tau ::= o \mid \sigma \mid \tau \to \tau \quad \text{where } \sigma \in \Sigma_t$$

For each type there is a countably infinite disjoint set of variables.

A *constant signature* Σ_c is a finite list of constant symbols with simple types $c:\tau_1, \ldots, c_m:\tau_m$. A *signature* Σ consists of a *type signature* Σ_t and a *constant signature* Σ_c.

The *terms* over the signature $\Sigma = \langle \Sigma_c, \Sigma_t \rangle$, ranged over by M, N, P, Q, R (possibly with indices), are defined by the following abstract syntax:

$$M ::= x \mid MN \mid \lambda x{:}\tau.M \mid c \mid M \Rightarrow N \mid \forall x{:}\tau.M \quad \text{where } c^\sigma \in \Sigma_c \text{ for some } \sigma$$

As usual, we denote by $M[N/x]$ capture-avoiding substitution. Terms are identified up-to α-conversion.

Remark 1. Primitive datatypes, such as natural numbers and lists, can be easily added to the metalanguage. For the sake of simplicity, however, we prefer to stick to the simplest theory of terms and types. Nevertheless, occasionally we use natural numbers in the following examples.

A *(typing) context* is a finite set of typing assertions over distinct variables (e.g. $\{x_1:\tau_1, x_2:\tau_2, \ldots, x_n:\tau_n\}$). Typing contexts are ranged over by Γ.

Typing judgements have the form $\Gamma \vdash_\Sigma M : \tau$, and they are inductively defined by the rules in Figure 1.

Terms of type o are the *propositions* of our logic. Terms of type $\tau \to o$ are called *predicates (over τ)*. As usual in HOL, all logical connectives can be defined in terms of $\forall$ and $\Rightarrow$. We list some definitions which will be used in the following:

$$\bot \triangleq \forall P{:}o.P \qquad \exists x{:}\tau.P \triangleq \neg\forall x{:}\tau.\neg p$$

$$\neg P \triangleq P \Rightarrow \bot \qquad M =^\tau N \triangleq \forall R{:}\tau \to o.(RM \Rightarrow RN)$$

$$P \vee Q \triangleq \neg P \Rightarrow Q \qquad P \wedge Q \triangleq \neg(P \Rightarrow \neg Q)$$

2.2 Encoding nominal algebras in Υ

The theory of Υ is expressive enough to represent faithfully any nominal algebra. The standard encoding methodology is based on the *higher-order abstract syntax* paradigm [7, 17], which can be succintely summarized as follows:

- object level names are represented by metalanguage variables;
- contexts are represented by higher-order terms, *i.e.* functions;
- binders are represented by constructors which take functions as arguments;
- contexts instantiation and capture-avoiding substitution are meta-level applications; hence, α-conversion is immediately inherited from the metalanguage.

Let $N = \langle V, I, C\rangle$ be a nominal algebra. The *signature for* N, $\Sigma(N)$, is defined as $\Sigma(N) \triangleq \langle V \cup I, \{c : \tau \mid c^\tau \in C\}\rangle$. Then, for each type $\tau \in V \cup I$ and for X stage in V, we define the *encoding map* ϵ^τ_X as follows:

- for $\upsilon \in V$, let $(n_i)_i$ be an enumeration of υ and $(x_i)_i$ be an enumeration of variables of type υ in Υ. Then, $\epsilon^\upsilon_X(n_i) \triangleq x_i$ for $n_i \in X_i$.
  Without loss of generality, we can identify objects of υ with variables of type υ, so that we can define $\epsilon^\upsilon_X(x) \triangleq x$.
- let $c^\alpha \in C$, where $\alpha = \tau_1 \times \cdots \times \tau_n \to \iota$, $\tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \tau'_i$ $(m_i \geq 0)$ and $\tau'_i \in V \cup I$. For $i = 1 \ldots n$, let Y_i be the stage whose components are $Y_{il} \triangleq X_l \uplus \{x_{ij} \mid \upsilon_{ij} = \upsilon_l, j = 1 \ldots m_i\}$, for $l = 1 \ldots k$. Let $t_i \in \mathcal{L}^{\tau'_i}_{Y_i}$. Then, we define $\epsilon^\iota(c^\alpha(t_1, \ldots, t_n)) \triangleq (c\ \lambda \boldsymbol{x}_1{:}\boldsymbol{\upsilon}_1.\epsilon^{\tau_1}_{Y_1}(t_1) \ldots \lambda \boldsymbol{x}_n{:}\boldsymbol{\upsilon}_n.\epsilon^{\tau_n}_{Y_n}(t_n))$, where $\lambda \boldsymbol{x}_i{:}\boldsymbol{\upsilon}_i$ is a shorthand for $\lambda x_{i1}{:}\upsilon_{i1} \ldots \lambda x_{im_i}{:}\upsilon_{im_i}$.

The canonical terms of Υ over $\Sigma(N)$ correspond faithfully to $\mathcal{L}(N)$:

Theorem 1 (Adequacy of encoding). *Let X be a stage in V, and let $\Gamma(X) \triangleq \{x : \upsilon_i \mid x \in X_i, i = 1 \ldots n\}$. For each type $\iota \in I$, the map ϵ^ι_X is a compositional bijection between $\mathcal{L}^\iota_X$ and the set of terms in $\beta\eta$-normal form of type ι in the context $\Gamma(X)$.*

Example 2. The nominal algebras of Example 1 can be encoded in Υ as follows:

- Untyped λ-calculus: $\Sigma(N_\lambda)_t = \upsilon, \Lambda$,
  $$\Sigma(N_\lambda)_c = var : \upsilon \to \Lambda, \boldsymbol{\lambda} : (\upsilon \to \Lambda) \to \Lambda, app : \Lambda \to \Lambda \to \Lambda$$
  For instance, $\epsilon^\Lambda_\emptyset(\boldsymbol{\lambda} x x x) = \boldsymbol{\lambda}\lambda x{:}\upsilon.(app\ (var\ x)\ (var\ x))$.
- SOL: $\Sigma(N_{SOL})_t = \upsilon, \upsilon', \iota, \phi$; $\Sigma(N_{SOL})_c = var : \upsilon \to \iota, var' : \upsilon' \to \phi, 0 : \iota, S : \iota \to \iota, =: \iota \to \iota \to \phi, \supset : \phi \to \phi \to \phi, \forall : (\upsilon \to \phi) \to \phi, \Lambda : (\upsilon' \to \phi) \to \phi$
- π-calculus: $\Sigma(N_\pi)_t = \upsilon, \iota$; $\Sigma(N_\pi)_c = 0 : \iota, \tau : \iota \to \iota, | : \iota \to \iota \to \iota, =: \upsilon \to \upsilon \to \iota \to \iota, \nu : (\upsilon \to \iota) \to \iota, in : \upsilon \to (\upsilon \to \iota) \to \iota, out : \upsilon \to \upsilon \to \iota \to \iota$

2.3 The Logical Framework Υ: The Logic

Our framework Υ is a full-blown higher order logic. The *logical derivation* judgement "$\Gamma; \Delta \vdash_\Sigma p$" expresses the fact that p derives from the set of propositions Δ in context Γ. Δ is a set of propositions $p_1, \ldots, p_n$ such that $\Gamma \vdash_\Sigma p_i : o$.

$$\frac{\Gamma;\Delta,p\vdash_\Sigma q}{\Gamma;\Delta\vdash_\Sigma p\Rightarrow q}\ (\Rightarrow\text{-I}) \qquad \frac{\Gamma\vdash_\Sigma p:o}{\Gamma;\Delta\vdash_\Sigma p\vee\neg p}\ (\text{LEM})$$

$$\frac{\Gamma;\Delta\vdash_\Sigma p\Rightarrow q \quad \Gamma;\Delta\vdash_\Sigma p}{\Gamma;\Delta\vdash_\Sigma q}\ (\Rightarrow\text{-E}) \qquad \frac{\Gamma,x:\tau\vdash_\Sigma M:\sigma \quad \Gamma\vdash_\Sigma N:\tau}{\Gamma;\Delta\vdash_\Sigma (\lambda x{:}\tau.M)N =^\sigma M[N/x]}\ (\beta)$$

$$\frac{\Gamma,x:\tau;\Delta\vdash_\Sigma p}{\Gamma;\Delta\vdash_\Sigma \forall x{:}\tau.p}x\notin FV(\Delta)\ (\forall\text{-I}) \qquad \frac{\Gamma\vdash_\Sigma M:\tau\to\sigma}{\Gamma;\Delta\vdash_\Sigma \lambda x{:}\tau.Mx =^{\tau\to\sigma} M}x\notin FV(M)\ (\eta)$$

$$\frac{\Gamma;\Delta\vdash_\Sigma \forall x{:}\tau.p \quad \Gamma\vdash_\Sigma M:\tau}{\Gamma;\Delta\vdash_\Sigma p[M/x]}\ (\forall\text{-E}) \qquad \frac{\Gamma,x:\sigma;\Delta\vdash_\Sigma M =^\tau N}{\Gamma;\Delta\vdash_\Sigma \lambda x{:}\sigma.M =^{\sigma\to\tau} \lambda x{:}\sigma.N}\ (\xi)$$

Fig. 2. Logical rules and axioms.

$$\frac{H_1 \quad \dots \quad H_n}{\Gamma;\Delta\vdash_\Sigma x\notin^\iota_\upsilon (c\ M_1\dots M_n)}c^{\tau_1\times\dots\times\tau_n\to\iota}\in C\qquad(\text{Notin}_c)$$

$$\text{where } H_i=\begin{cases}\Gamma;\Delta\vdash_\Sigma \neg(x=^\upsilon M_i) & \text{if } \tau_i=\upsilon\\ \Gamma,\Gamma_i;\Delta,\Delta_i\vdash_\Sigma x\notin^{\iota'}_\upsilon (M_i\ y_1\dots y_{m_i}) & \text{if } \tau_i=\upsilon_{i1}\times\dots\times\upsilon_{im_i}\to\iota'\end{cases}$$

$$\Gamma_i=y_1:\upsilon_{i1},\dots,y_{m_i}:\upsilon_{im_i}\qquad \Delta_i=\{\neg(x=^\upsilon y_j)\mid \upsilon_j=\upsilon, j=1\dots m_i\}$$

Fig. 3. Rules for non-occurrence predicates.

The system for deriving judgements consists of a set of *logical* rules and axioms and a set of axioms representing the *Theory of Contexts*. The logical rules and axioms (see Figure 2) consist of a natural deduction-style system for classical higher-order logic, with $\beta\eta\xi$-equality.

Before introducing the Theory of Contexts, we define *non-occurrence* $\notin^\iota_\upsilon$: $\upsilon\to\iota\to o$ for each $\upsilon\in V$ and $\iota\in I$. The intuitive meaning of a proposition "$x\notin^\iota_\upsilon M$" (read "x not in M") is "the name x (of type υ) does not appear free in M (of type ι)." (The index $_\upsilon$ will be dropped, when clear from the context.) Since we have higher-order logic, these predicates can be defined by means of higher-order quantifications and monotone operators over predicates, as in [1,10]:

$$x\notin^\iota_\upsilon M \triangleq \forall P{:}\upsilon\to\iota\to o.(\forall y{:}\upsilon.\forall N{:}\iota.(T_{\notin^\iota_\upsilon}\ P\ y\ N)\Rightarrow(P\ y\ N))\Rightarrow(P\ x\ M)$$

where $T_{\notin^\iota_\upsilon}:(\upsilon\to\iota\to o)\to(\upsilon\to\iota\to o)$ is a suitable monotone operator defined on the syntax of the language, i.e., on the constructors in C. An explicit definition of these operators, although straightforward, would be quite cumbersome, especially in the case of languages with mutually defined syntactic sorts. Thus for the sake of simplicity we give an implicit definition of the underlying operators by means of a set of "derivation rules" for $\notin^\iota_\upsilon$, as described in Figure 3. It should be clear, however, that these rules are *derivable* from the impredicative definition of the non-occurrence predicates. From a practical point of view, moreover, a rule-based definition is closer to the approach which would be used in proof assistants featuring inductive predicates, as it has been done in [10] using Coq.

Proposition 1 (Adequacy of $\notin^\tau_\upsilon$). *For all Γ contexts, $(x:\upsilon)\in\Gamma$ and M such that $\Gamma\vdash_\Sigma M:\iota$, we have: $\Gamma;\emptyset\vdash_\Sigma x\notin^\tau_\upsilon M$ iff $x\notin FV(M)$*

$$\frac{\Gamma \vdash_\Sigma P : \iota}{\Gamma; \Delta \vdash_\Sigma \exists x{:}\upsilon.x \notin P} \quad (Unsat_\iota^\upsilon)$$

$$\frac{\Gamma \vdash_\Sigma P : \upsilon \to \tau \quad \Gamma \vdash_\Sigma Q : \upsilon \to \tau \quad \Gamma \vdash_\Sigma x : \upsilon}{\Gamma; \Delta, x \notin^{\upsilon\to\tau} P, x \notin^{\upsilon\to\tau} Q, (P\ x) =^\tau (Q\ x) \vdash_\Sigma P =^{\upsilon\to\tau} Q} \quad (Ext_\upsilon^\tau)$$

$$\frac{\Gamma \vdash_\Sigma P : \tau \quad \Gamma \vdash_\Sigma x : \upsilon}{\Gamma; \Delta \vdash_\Sigma \exists Q{:}\upsilon \to \tau.x \notin^{\upsilon\to\tau} Q \wedge P =^\tau (Q\ x)} \quad (\beta\_exp_\upsilon^\tau)$$

where $\tau = \upsilon_{i_1} \to \cdots \to \upsilon_{i_k} \to \iota$

Fig. 4. Axiom schemata for the Theory of Contexts.

Proof. By induction on the derivation ($\Rightarrow$), and on the syntax of M ($\Leftarrow$). □

Non-occurrence predicates can be lifted to contexts, that is terms of type $\upsilon_{i_1} \to \cdots \to \upsilon_{i_k} \to \iota$:

$$x \notin_\upsilon^{\upsilon\to\tau} M \triangleq \forall y{:}\upsilon.\neg(x =^\upsilon y) \Rightarrow x \notin_\upsilon^\tau (M\ y)$$
$$x \notin_\upsilon^{\upsilon'\to\tau} M \triangleq \forall y{:}\upsilon'.x \notin_\upsilon^\tau (M\ y) \qquad (\upsilon \neq \upsilon')$$

Now we can introduce the second set of axioms (Figure 4). This is the real core of the Theory of Contexts. We do not give any informal motivation since these axioms reflect rather natural properties of contexts, when these are rendered by λ-abstractions. They are crucial for reasoning on terms in higher-order abstract syntax, see Theorem 3 and case studies (Section 4).

Remark 2. Our experience in encoding and formally proving metatheoretical properties of nominal algebras indicates that full classical logic is not strictly needed. Actually we could drop rule LEM in Figure 2 and simply assume that either Leibniz equality over names is decidable or occurrence predicates of names in terms are decidable. Indeed, this is the approach we adopted in [10]. More specifically, the two above axioms are rendered in Υ as follows:

$$\frac{\Gamma \vdash_\Sigma x : \upsilon \quad \Gamma \vdash_\Sigma y : \upsilon}{\Gamma; \Delta \vdash_\Sigma x =^\upsilon y \vee x \neq^\upsilon y} \ (\mathrm{LEM}_{=^\upsilon}) \qquad \frac{\Gamma \vdash_\Sigma x : \upsilon \quad \Gamma \vdash_\Sigma P : \iota}{\Gamma; \Delta \vdash_\Sigma x \notin_\upsilon^\iota P \vee \neg(x \notin_\upsilon^\iota P)} \ (\mathrm{LEM}_{\notin_\upsilon^\iota})$$

It is worth noticing that $\mathrm{LEM}_{=^\upsilon}$ derives directly from $\mathrm{LEM}_{\notin_\upsilon^\iota}$. On the converse, $\mathrm{LEM}_{\notin_\upsilon^\iota}$ can be derived from $\mathrm{LEM}_{=^\upsilon}$ using $Unsat_\iota^\upsilon$ and by induction both over plain terms and over contexts, using the principles Ind^ι, $Ind^{\upsilon\to\iota}$ (Section 3.1.)

Remark 3. In [10] the Theory of Contexts is enriched by another rather useful axiom stating the monotonicity of $\notin_\upsilon^\iota$:

$$\frac{\Gamma \vdash_\Sigma x : \upsilon \quad \Gamma \vdash_\Sigma y : \upsilon \quad \Gamma \vdash_\Sigma p : \upsilon \to \iota}{\Gamma; \Delta, x \notin_\upsilon^\iota (p\ y) \vdash_\Sigma x \notin_\upsilon^{\upsilon\to\iota} p} \quad (\mathrm{MON}_{\notin_\upsilon^\iota})$$

Recently, we discovered that the latter is indeed derivable from $Unsat_\iota^\upsilon$, $\mathrm{LEM}_{\notin_\upsilon^\iota}$ and Ind^ι. Another possibility of deriving $\mathrm{MON}_{\notin_\upsilon^\iota}$ is to exploit $Ind^{\upsilon\to\iota}$ without any other axioms, i.e., to reason inductively on the structure of the context p.

2.4 Properties of Υ

One upmost concern is soundness:

Theorem 2. *For all nominal algebras N, the framework Υ over the signature $\Sigma(N)$ is sound, in the sense that for all Γ, it is not the case that $\Gamma;\emptyset \vdash_{\Sigma(N)} \bot$.*

The proof of this theorem relies on the construction of a model based on functor categories, following the approach presented in [8]. The construction is quite complex; for further details, we refer the reader to [1, 20].

It would be interesting to investigate and formulate more expressive soundness results, which could substantiate the intended meaning of our framework, possibly based on this model. In this paper we discuss solely an important property of Υ which motivates the very Theory of Contexts and can be viewed as a first result in this direction. Let $\Gamma \vdash_\Sigma p : \upsilon \to o$; since terms are taken up-to α-equivalence, we would like the following two rules to be derivable in Υ:

$$\frac{\Gamma;\Delta \vdash_\Sigma \forall y{:}\upsilon.y \not\in^{\upsilon\to o} p \Rightarrow (p\ y)}{\Gamma;\Delta \vdash_\Sigma \exists y{:}\upsilon.y \not\in^{\upsilon\to o} p \wedge (p\ y)}\ (\forall\exists) \qquad \frac{\Gamma;\Delta \vdash_\Sigma \exists y{:}\upsilon.y \not\in^{\upsilon\to o} p \wedge (p\ y)}{\Gamma;\Delta \vdash_\Sigma \forall y{:}\upsilon.y \not\in^{\upsilon\to o} p \Rightarrow (p\ y)}\ (\exists\forall)$$

Indeed, we have that

Theorem 3. *In Υ: $\forall\exists$ is derivable, and $\exists\forall$ is admissible.*

Proof. (Idea.) For $\forall\exists$: some first-order reasoning and an application of $Unsat^{\upsilon}_{\iota}$.

For $\exists\forall$: after some first-order reasoning using Ext^{τ}_{υ}, it is easy to see that this rule is implied by the following *fresh renaming* rule:

$$\frac{\Gamma, x:\upsilon;\Delta, x \not\in^{\upsilon\to o} p \vdash_\Sigma (p\ x)}{\Gamma, y:\upsilon;\Delta, y \not\in^{\upsilon\to o} p \vdash_\Sigma (p\ y)} x, y \not\in FV(\Delta) \qquad \text{(Ren)}$$

Rule Ren is admissible in our system, because there is no way for a predicate in Υ to discriminate between two fresh names. Thus, a derivation for $(p\ y)$ can be readily obtained by mimicking that for $(p\ x)$ by replacing every free occurrence of x by y. □

Using Ext^{τ}_{υ} and $\beta\_exp^{\tau}_{\upsilon}$, one can prove that the rule schema $\exists\forall$ is indeed *derivable* in Υ and many of its extensions, for most specific predicates of interest. This is the case, for instance, of the transition relation and strong (late) bisimilarity of π-calculus in Coq [10].

Rule Ren is only admissible because there can be non-standard notions of "predicates" which do not satisfy rule Ren and still are consistent with respect to the Theory of Contexts. An example can be constructed from a logical framework with a distinct type of names $\bar{\upsilon}$, denoting the sum of all classes in the names base. Then, in a nominal algebra with at least two names sets, it would be possible to define a predicate over $\bar{\upsilon}$ which discriminates "red" bound names from "black" bound names, thus invalidating rule Ren.

The type theory of Υ satisfies all the usual properties of simply typed λ-calculi:

Theorem 4. *Let Γ be a context, M, N be terms and τ, σ be types.*

- *(Uniqueness of type) If $\Gamma \vdash_\Sigma M : \tau$ and $\Gamma \vdash_\Sigma M : \sigma$, then $\tau = \sigma$*
- *(Subject reduction) If $\Gamma \vdash_\Sigma M : \tau$ and $\Gamma; \emptyset \vdash_\Sigma M =^\tau N$, then $\Gamma \vdash_\Sigma N : \tau$*
- *(Normal form) If $\Gamma \vdash_\Sigma M : \tau$, then there exists N in canonical form such that $\Gamma; \emptyset \vdash_\Sigma M =^\tau N$*
- *(Church-Rosser) If $\Gamma; \Delta \vdash_\Sigma M =^\tau N_1$ and $\Gamma; \Delta \vdash_\Sigma M =^\tau N_2$ then there exists N such that $\Gamma; \Delta \vdash_\Sigma N_1 =^\tau N$ and $\Gamma; \Delta \vdash_\Sigma N_2 =^\tau N$*

3 Induction and Recursion in $\mathcal{Y}$

In this section we face the problem of defining relations and functions by structural induction and recursion, in $\mathcal{Y}$, and in particular over higher-order types. This very possibility is one of the main properties of the frameworks in [6, 5, 4].

As originally pointed out in [8] for the case of λ-calculus, a rather surprising tradeoff of our natural framework for treating contexts is the following:

Proposition 2. *The Axiom of Unique Choice*

$$\frac{\Gamma \vdash R : \sigma \to \tau \to o \quad \Gamma, a : \sigma; \Delta \vdash \exists! b : \tau.(R\ a\ b)}{\Gamma; \Delta \vdash \exists f : \sigma \to \tau. \forall a : \sigma.(R\ a\ (f\ a))} \qquad (\mathrm{AC!}_{\sigma,\tau})$$

is inconsistent with the Theory of Contexts.

Proof. Let us consider the case of the π-calculus encoding (see N_π in Example 1), then, by $Unsat^\upsilon_\iota$, we can infer the existence of two fresh names u', v'; hence, we can define the term $R \triangleq \lambda u : \upsilon.\lambda q : \iota.\lambda x : \upsilon.\lambda p : \upsilon.(x =^\upsilon u \wedge p =^\iota 0) \vee (\neg x =^\upsilon u \wedge p =^\iota q)$. It is easy to show that, for all $p' : \iota$, $(R\ u'\ p') : \upsilon \to \iota \to o$ is a functional binary relation. At this point we can prove, by means of Ext^ι_υ and $\mathrm{AC!}_{\upsilon,\iota}$, that the proposition $\forall p : \iota.p =^\iota 0$ holds; Indeed, from $\mathrm{AC!}_{\upsilon,\iota}$ we can deduce the existence of a function $f : \upsilon \to \iota$ such that, for all $x : \upsilon$, $((R\ u'\ p)\ x\ (f\ x))$ holds. Hence, by Ext^ι_υ, we can prove that $f =^{\upsilon\to\iota} \lambda x : \upsilon.p$ because for any fresh name w we have that $(f\ w) =^\iota p$. Then we have that, for all names y, $(f\ y) =^\iota ((\lambda x : \upsilon.p)\ y) =^\iota p$ holds, whence the thesis, since $(f\ u') = 0$.

At this point the contradiction follows because, as a special case, we have that $0|0 = 0$ while ι is an inductive type (the constructors are injective). □

As a consequence of Proposition 2, there are (recursive) functions which cannot be defined as such in $\mathcal{Y}$, but which can nevertheless be described as functional (inductively defined) relations. We will elaborate more on this in Remark 4.

A rather powerful theory of recursion can nonetheless be obtained also within $\mathcal{Y}$, following Hofmann [8], exploiting in particular the initial algebra property of the tripos model [1].

$$\frac{\{\Gamma, \Gamma_c; \Delta, \Delta_c \vdash_\Sigma (P_\iota\ (c\ x_1 \dots x_n)) \mid c^{\tau_1 \times \dots \times \tau_n \to \iota} \in C\}}{\Gamma; \Delta \vdash_\Sigma \forall x{:}\iota.(P_\iota\ x)} \quad (Ind^\iota)$$

where $\tau_i = \upsilon_{i1} \times \dots \times \upsilon_{im_i} \to \iota_i$

$$\Gamma_c \triangleq x_1 : \tau_1', \dots, x_n : \tau_n' \qquad (\tau_i' = \upsilon_{i1} \to \dots \to \upsilon_{im_i} \to \iota_i)$$

$$\Delta_c \triangleq \{\forall y_1{:}\upsilon_{i1}. \dots . \forall y_{m_i}{:}\upsilon_{im_i}.(P_{\iota_i}\ (x_i\ y_1 \dots y_{m_i})) \mid i = 1, \dots, n, \tau_i \notin V\}$$

Fig. 5. First-order induction principle.

3.1 First-Order and Higher-Order Induction

The usual induction principles can be extended naturally to terms possibly containing higher-order terms. In Figure 5 we give the definition of a induction schema for any nominal algebra. In case terms of a syntactic category contain terms of other categories, this scheme turns out to be a mutual induction principle. In this case several inductive properties, one for each syntactic category, are mutually defined. This schema is a direct generalization of those obtained in Coq by Scheme ... Induction [11].

Example 3. The induction principle over terms of π-calculus of the previous examples is the following:

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash (P_\iota\ 0) \qquad \Gamma, x : \iota; \Delta, (P_\iota\ x) \vdash (P_\iota\ (\tau x)) \\ \Gamma, x_1 : \iota, x_2 : \iota; \Delta, (P_\iota\ x_1), (P_\iota\ x_2) \vdash (P_\iota\ (x_1 | x_2)) \\ \Gamma, x_1 : \upsilon, x_2 : \upsilon, x_3 : \iota; \Delta, (P_\iota\ x_3) \vdash (P_\iota\ [x_1 = x_2] x_3) \\ \Gamma, x : \upsilon \to \iota; \Delta, \forall y{:}\upsilon.(P_\iota\ (x\ y)) \vdash (P_\iota\ \nu x) \\ \Gamma, x_1 : \upsilon, x_2 : \upsilon \to \iota; \Delta, \forall y{:}\upsilon.(P_\iota\ (x_2\ y)) \vdash (P_\iota\ (in\ x_1\ x_2)) \\ \Gamma, x_1 : \upsilon, x_2 : \upsilon, x_3 : \iota; \Delta, (P_\iota\ x_3) \vdash (P_\iota\ (out\ x_1\ x_2\ x_3))\end{array}}{\Gamma; \Delta \vdash \forall x{:}\iota.(P_\iota\ x)}$$

The induction principles can be consistently axiomatized also for higher-order terms, i.e. contexts. For the sake of simplicity, we present the general forms of induction principles for simple contexts, that is terms of type $\upsilon \to \iota$ (Figure 6). Nevertheless, the principles can be generalized to any number of abstractions; see [1] for an example and for the semantic justification.

Notice that the terms t_i's are non deterministically defined. In the case that τ_i is exactly υ, both the "x_i" and "x" cases apply. This means that two premises, one for each possible case, have to be taken. Hence, for each constructor c of type $\tau_1 \times \dots \times \tau_n \to \iota$, if υ occurs k times in the sequence $\tau_1, \dots, \tau_n$, then in the rule there are 2^k premises. The following example should make this clear.

Example 4. The induction principle over contexts of λ-calculus is the following:

$$\frac{\begin{array}{c}\Gamma, x_1 : \upsilon; \Delta \vdash (P\ \lambda x{:}\upsilon.(var\ x_1)) \qquad \Gamma, x_1 : \upsilon; \Delta \vdash (P\ \lambda x{:}\upsilon.(var\ x)) \\ \Gamma, M_1{:}\upsilon \to \Lambda, M_2{:}\upsilon \to \Lambda; \Delta, (P\ M_1), (P\ M_2) \vdash (P\ \lambda x{:}\upsilon.(app\ (M_1\ x)\ (M_2\ x))) \\ \Gamma, M_1 : \upsilon \to \upsilon \to \Lambda; \Delta, \forall y_1{:}\upsilon.(P\ \lambda x{:}\upsilon.(M_1\ x\ y_1)) \vdash (P\ \lambda x{:}\upsilon.(\boldsymbol{\lambda}(M_1\ x)))\end{array}}{\Gamma; \Delta \vdash \forall M{:}\upsilon \to \Lambda.(P\ M)}$$

$$\frac{\{\Gamma, \Gamma_c; \Delta, \Delta_c \vdash_\Sigma (P_{\upsilon\to\iota}\ \lambda x{:}\upsilon.(c\ t_1 \dots t_n)) \mid c^{\tau_1\times\cdots\times\tau_n\to\iota} \in C\}}{\Gamma; \Delta \vdash_\Sigma \forall M{:}\upsilon \to \iota.(P_{\upsilon\to\iota}\ M)} \qquad (Ind^{\upsilon\to\iota})$$

where $\tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \iota_i$ and

$$\Gamma_c \triangleq \{M_i : \upsilon \to \tau_i' \mid \tau_i \notin V\} \cup \{x_i : \tau_i \mid \tau_i \in V\} \quad (\tau_i' = \upsilon_{i1} \to \cdots \to \upsilon_{im_i} \to \iota_i)$$

$$\Delta_c \triangleq \{\forall y_1{:}\upsilon_{i1} \dots \forall y_{m_i}{:}\upsilon_{im_i}.(P_{\upsilon\to\iota_i}\ \lambda x{:}\upsilon.(M_i\ x\ y_1 \dots y_{m_i})) \mid i = 1, \dots, n, \tau_i \notin V\}$$

$$t_i \triangleq \begin{cases} (M_i\ x) & \text{if } \tau_i \notin V \\ x_i & \text{if } \tau_i \in V, \tau_i = \upsilon \\ x_i \text{ or } x & \text{if } \tau_i = \upsilon \end{cases}$$

Fig. 6. Higher-order induction principle.

This principle is stronger than the one provided by Coq. In fact, Coq does not recognize that $(M_1\ x)$ is structurally smaller than $\lambda x : \upsilon.(\boldsymbol{\lambda}\ (M_1\ x))$.

3.2 First-Order and Higher-Order Recursion

Despite the drawback we mentioned earlier, many interesting functions can be defined by recursion anyway. Υ can be extended with *recursion operators* or simply *recursors*, which smoothly generalize the usual ones for plain datatypes to HOAS types, and even higher-order types.

Definition 4. *Let $N = \langle V, I, C\rangle$ be a nominal algebra. Let $\iota \in I$, τ be a type in $\Sigma(N)$, and let Γ be a context. An* elimination scheme over τ (in Γ) *is a family of terms $F^\tau = \{f_c \mid c^\alpha \in C\}$ such that for $c^{\tau_1\times\cdots\times\tau_n\to\iota} \in C$, we have*

$$\Gamma \vdash f_c : \tau_1' \to \cdots \to \tau_n' \to \tau$$

where $\tau_i' \triangleq \begin{cases} \tau_i & \text{if } \tau_i \in V \\ \upsilon_{i1} \to \cdots \to \upsilon_{im_i} \to \tau & \text{if } \tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \iota_i \end{cases}$

Let F^τ be an elimination scheme in Γ. For each ι in I, we introduce in the language a new symbol, $\hat{F}_\iota^\tau$, called the *F-defined recursive map over ι*. In Figure 7 we give the typing rules for each recursive map, and the equivalence rules for each constructor. This schema naturally generalizes the usual one to the case of mutually defined recursive maps, and to terms possibly containing higher-order terms. This schema is validated by the model of the Theory of Contexts [1]. This recursion schema allows to define many functions on the syntax of terms, like in the following Example.

Example 5. In the signature $\Sigma(N_\lambda)$ of the previous example, we will define, by recursion, a map which counts the number of λ-abstractions in a λ-term. Let $F^{nat} \triangleq \{f_{var}, f_{app}, f_{\boldsymbol{\lambda}}\}$ where $f_{var} \triangleq 0$, $f_{app} \triangleq \lambda n{:}nat.\lambda m{:}nat.n + m$, $f_{\boldsymbol{\lambda}} \triangleq \lambda g{:}\upsilon \to nat.(g\ z) + 1$. Then, F^{nat} is an elimination scheme in $z : \upsilon$, and in any context Γ containing $z : \upsilon$, we have that $\Gamma \vdash \hat{F}_\Lambda^{nat} : \Lambda \to nat$ and the following

$$\frac{F^\tau \text{ elimination scheme over } \tau \text{ in } \Gamma}{\Gamma \vdash_\Sigma \hat{F}^\tau_\iota : \iota \to \tau} \qquad (\hat{F}^\tau_\iota)$$

$$\frac{\Gamma \vdash_\Sigma \hat{F}^\tau_\iota : \iota \to \tau \quad \Gamma \vdash_\Sigma t_1{:}\tau_1 \quad \ldots \quad \Gamma \vdash_\Sigma t_n{:}\tau_n}{\Gamma; \Delta \vdash_\Sigma (\hat{F}^\tau_\iota\ (c\ t_1 \ldots t_n)) =^\tau (f_c\ M_1 \ldots M_n)} \qquad (\hat{F}^\tau_\iota\text{-}eq_c)$$

where $c^{\tau_1 \times \cdots \times \tau_n \to \iota} \in C$, $f_c \in F^\tau$ and for $i = 1 \ldots n$:

$$M_i \triangleq \begin{cases} \lambda x_{i1}{:}\upsilon_{i1} \ldots \lambda x_{im_i}{:}\upsilon_{im_i}.(\hat{F}^\tau_{\iota_i}\ (t_i\ x_{i1} \ldots x_{im_i})) & \text{if } \tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \iota_i \\ t_i & \text{if } \tau_i \in V \end{cases}$$

Fig. 7. First-order recursion typing and equivalence rules

are derivable:

$$\frac{\Gamma \vdash x : \upsilon}{\Gamma; \Delta \vdash \hat{F}^{nat}_\Lambda(var\ x) =^{nat} 0} \qquad \frac{\Gamma \vdash M : \upsilon \to \Lambda}{\Gamma; \Delta \vdash \hat{F}^{nat}_\Lambda(\boldsymbol{\lambda} M) =^{nat} \hat{F}^{nat}_\Lambda(M\ z) + 1}$$

$$\frac{\Gamma \vdash M : \Lambda \quad \Gamma \vdash N : \Lambda}{\Gamma; \Delta \vdash \hat{F}^{nat}_\Lambda(app\ M\ N) =^{nat} \hat{F}^{nat}_\Lambda(M) + \hat{F}^{nat}_\Lambda(N)}$$

Recursion principles can be consistently axiomatized for higher-order terms, i.e. contexts. For the sake of simplicity, we present the general forms of recursion principles for simple contexts, that is terms of type $\upsilon \to \iota$. Nevertheless, the principles can be generalized to any number of abstractions; see [1] for an example and for the semantic justification of the principles.

Before giving the definitions and the rules, we need some notation. Let $\alpha = \tau_1 \times \cdots \times \tau_n \to \iota$ be an arity of constructor. Let $\upsilon \in V$ and let $1 \le i_1 \le \cdots \le i_k \le n$ $(k \ge 0)$ be the indices such that $\tau_{i_j} = \upsilon$. Let $L(\alpha) \triangleq \{0,1\}^k$ be the set of binary strings of length k, which we call the *labels for* α. (Thus, $|L(\alpha)| = 2^k$.) For $j = 1 \ldots k$, the j-th component of a label l is denoted by l_{i_j}, that is it has the same index of the occurrence of υ in $\tau_1 \ldots \tau_n$ it refers to. Finally, we denote by $l \bullet (\tau'_1 \to \cdots \to \tau'_n \to \tau)$ the type obtained from $\tau'_1 \to \cdots \to \tau'_n \to \tau$ by eliminating τ'_{i_j} if $l_{i_j} = 0$.

Definition 5. *Let $\iota \in I$, $\upsilon \in V$, in $\Sigma(N)$, and let Γ be a context. A υ-*elimination scheme over τ (in Γ) *is a family of terms $F^\tau = \{f^l_c \mid c^\alpha \in C, l \in L(\alpha)\}$ such that for $c^{\tau_1 \times \cdots \times \tau_n \to \iota} \in C$ and $l \in L(\tau_1 \times \cdots \times \tau_n \to \iota)$, we have*

$$\Gamma \vdash f^l_c : l \bullet (\tau'_1 \to \cdots \to \tau'_n \to \tau)$$

where $\tau'_i \triangleq \begin{cases} \upsilon & \text{if } \tau_i = \upsilon \\ \upsilon_{i1} \to \cdots \to \upsilon_{im_i} \to \tau & \text{if } \tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \iota_i \end{cases}$

Hence, for each constructor c^α, there are $|L(\alpha)|$ terms in F^τ.

Let F^τ be a υ-elimination scheme in Γ. For each ι in I, we introduce in the language a new symbol, $\hat{F}^\tau_\iota$, called the *F-defined recursive map over $\upsilon \to \iota$*. In Figure 8 we give the typing rules for each recursive map, and the equivalence

$$\frac{F^\tau \text{ } \upsilon\text{-elimination scheme over } \tau \text{ in } \Gamma}{\Gamma \vdash_\Sigma \hat{F}^\tau_\iota : (\upsilon \to \iota) \to \tau} \qquad (\hat{F}^\tau_\iota)$$

$$\frac{\Gamma \vdash_\Sigma \hat{F}^\tau_\iota : (\upsilon \to \iota) \to \tau \quad \{\Gamma \vdash_\Sigma t_i : \upsilon \to \tau_i \mid \tau_i \notin V\} \quad \{\Gamma \vdash_\Sigma y_i : \tau_i \mid \tau_i \in V\}}{\Gamma; \Delta \vdash_\Sigma (\hat{F}^\tau_\iota \ \lambda x{:}\upsilon.(c\ N_1 \dots N_n)) =^\tau (f^l_c\ M_1 \dots M_n)} \qquad (\hat{F}^\tau_\iota\_eq^l_c)$$

where $c^{\tau_1 \times \cdots \times \tau_n \to \iota} \in C$, $l \in L(\tau_1 \times \cdots \times \tau_n \to \iota)$, $f^l_c \in F^\tau$, and for $i = 1 \dots n$:

$$N_i = \begin{cases} (t_i\ x) & \text{if } \tau_i \notin V \\ y_i & \text{if } \tau_i \in V \text{ and } (\tau_i \neq \upsilon \text{ or } l_i = 1) \\ x & \text{if } \tau_i = \upsilon \text{ and } l_i = 0 \end{cases}$$

$$M_i = \begin{cases} \lambda x_{i1}{:}\upsilon_{i1} \dots \lambda x_{im_i}{:}\upsilon_{im_i}.(\hat{F}^\tau_{\iota_i}\ \lambda x{:}\upsilon.(t_i\ x\ x_{i1} \dots x_{im_i})) & \\ \qquad\qquad \text{if } \tau_i = \upsilon_{i1} \times \cdots \times \upsilon_{im_i} \to \iota_i & \\ y_i \qquad \text{if } \tau_i \in V \text{ and } (\tau_i \neq \upsilon \text{ or } l_i = 1) & \\ (\mathit{nothing}) \qquad \text{if } \tau_i = \upsilon \text{ and } l_i = 0 & \end{cases}$$

Fig. 8. Higher-order recursion typing and equivalence rules.

rules for each term in F^τ. This schema is validated by the categorical model of the Theory of Contexts [1].

Example 6. In the signature $\Sigma(N_\lambda)$, we will define, by higher-order recursion, the substitution $\cdot[N/\cdot]$ for a given λ-term such that $\Gamma \vdash N : \Lambda$. The three sets of labels are $L(\upsilon \to \Lambda) = \{0, 1\}$, $L(\Lambda \to \Lambda \to \Lambda) = L((\upsilon \to \Lambda) \to \Lambda) = \{\langle\rangle\}$. Thus, let $F^\Lambda \triangleq \{f^0_{var}, f^1_{var}, f_{app}, f_{\boldsymbol{\lambda}}\}$ where $f^0_{var} \triangleq N$, $f^1_{var} \triangleq var$, $f_{app} \triangleq app$, $f_{\boldsymbol{\lambda}} \triangleq \boldsymbol{\lambda}$. Then, F^Λ is a υ-elimination schema in Γ, such that $\Gamma \vdash \hat{F}^\Lambda_\Lambda : (\upsilon \to \Lambda) \to \Lambda$ and the following are derivable:

$$\frac{-}{\Gamma; \Delta \vdash \hat{F}^\Lambda_\Lambda(var) =^\Lambda N} \qquad \frac{\Gamma \vdash y : \upsilon}{\Gamma; \Delta \vdash \hat{F}^\Lambda_\Lambda(\lambda x{:}\upsilon.(var\ y)) =^\Lambda (var\ y)}$$

$$\frac{\Gamma \vdash M : \upsilon \to \upsilon \to \Lambda}{\Gamma; \Delta \vdash \hat{F}^\Lambda_\Lambda(\lambda x{:}\upsilon.\boldsymbol{\lambda}(M\ x)) =^\Lambda \boldsymbol{\lambda}\lambda x{:}\upsilon.\hat{F}^\Lambda_\Lambda(M\ x)}$$

$$\frac{\Gamma \vdash M_1 : \upsilon \to \Lambda \quad \Gamma \vdash M_2 : \upsilon \to \Lambda}{\Gamma; \Delta \vdash \hat{F}^\Lambda_\Lambda(\lambda x{:}\upsilon.(app\ (M_1\ x)\ (M_2\ x)) =^\Lambda (app\ \hat{F}^\Lambda_\Lambda(M_1)\ \hat{F}^\Lambda_\Lambda(M_2))}$$

Hence, for any M term and x variable, the term $\hat{F}^\Lambda_\Lambda(\lambda x{:}\upsilon.M)$ is equal to the term obtained from M by replacing every free occurrence of x by N.

Remark 4. There are functions that we cannot define in Υ. The reason is that, Υ does not allow to define functions whose definitions need freshly generated names, since there is no means for generating a "fresh name" at the term level, while we can use $Unsat^\upsilon_\tau$ for generating fresh names at the logical level. Nevertheless, n-ary functions of this kind can be represented in Υ as $(n+1)$-ary relations, as in the next Example.

Example 7. Let us consider the function $count : \Lambda \to nat$ which takes as argument a term M of type Λ and returns the number of occurrences of free variables occurring in M. The corresponding elimination scheme over nat should be $f_{var} \triangleq \lambda x : \upsilon.1$, $f_{app} \triangleq \lambda n : nat.\lambda n' : nat.n + n'$, $f_{\boldsymbol{\lambda}} \triangleq \lambda g : \upsilon \to nat.(g\ z)\dot{-}1$, where $\dot{-}1$ denotes the predecessor function over natural numbers. However, the above definition cannot be expressed in Υ since the fresh name z, needed in the definition of $f_{\boldsymbol{\lambda}}$, is not definable. We do not have a mechanism working at the level of datatypes for generating fresh names on the spot, like Gabbay and Pitts' *fresh* operator [6]. It is straightforward that, in the presence of such a fresh operator, $f_{\boldsymbol{\lambda}}$ can be defined as $\lambda g : \upsilon \to nat.fresh\ z.(g\ z)\dot{-}1$. However, we can represent $f_{\boldsymbol{\lambda}}$ as a binary relation $R_{\boldsymbol{\lambda}} : (\upsilon \to nat) \to nat \to o$ defined as

$$R_{\boldsymbol{\lambda}}(g, n) \triangleq \exists z : nat.z \not\in^{\upsilon \to nat} g \wedge (g\ z)\dot{-}1 =^{nat} n$$

the existence of the fresh variable z being granted by $Unsat^{\upsilon}_{nat}$. Hence, the *fresh* operator can be mimicked at the logical level by our $Unsat^{\upsilon}_{\iota}$ axiom scheme.

4 Case studies

In order to prove the usability and expressiveness of our axiomatic Theory of Contexts, we carried out several case studies about metatheoretical reasoning on HOAS-based encodings. The first case study, which indeed yielded the first version of our axiomatization, is the encoding of Milner's π-calculus as presented in [16]. We refer for more details to [10]; here we will only remark that the Theory of Contexts allowed us to formally derive in Coq a substantial chapter of the metatheory of strong late bisimilarity. For instance, the following property

[16, Lemma 6] If $P \dot{\sim} Q$ and $w \notin fn(P, Q)$, then $P\{w/x\} \dot{\sim} Q\{w/x\}$.

has been formally derived, using all axioms of the Theory of Contexts.

Another substantial case study concerned the untyped and simply typed λ-calculus in Coq [15]. A set of important metatheoretical results has been formally proved by means of the Theory of Contexts, such as determinism and totality of substitution:

For all M, N, x, there exists exactly one M' such that $M' = M[N/x]$.

In this development, substitution has been defined as a relation between contexts and terms, and the proof of totality relies on higher-order induction over contexts. Other properties which have been proved include determinism and totality of big-step semantics, subject reduction and equivalence between small-step and big-step semantics. A similar case study has been carried out for First Order Logic, and on a smaller scale for a λ-calculus with explicit substitutions.

Currently there is work in progress on more complex process algebras, namely the spi calculus and the ambient calculus. These case studies are quite challenging for testing the expressivity of our axiomatization, since they provide a notion of substitution of terms for names, while the original π-calculus only relies on substitution of names for names. Some of the modal logics for ambient calculi are troublesome also because they introduce distinct sets for names and variables.

5 Comparison with related work and concluding remarks

The Theory of Contexts and Isabelle/HOL. The Theory of Contexts can be used in many different logical frameworks in order to reason on higher-order abstract syntax. A HOAS-based encoding of the syntax of π-calculus processes in Isabelle/HOL is given in [19]. The axioms $MON_{\not\in^\iota_\upsilon}$, Ext^ι_υ and $\beta\_exp^\iota_\upsilon$ are formally derived there from well-formedness predicates. The proof of the monotonicity of the occurrence predicate $\not\in^\iota_\upsilon$ is straightforward, since this is not defined independently, but simply as the negation of $\in^\iota_\upsilon$. It can be formally proved equivalent to the constructive one, however, by means of $Unsat^\upsilon_\iota$ and Ind^ι and $\mathrm{LEM}_{=^\upsilon}$. The proofs of extensionality and β-expansion rely heavily on the fact that Isabelle/HOL implements an extensional equality. These proofs cannot be mimicked in COQ.

The FM approach. Gabbay and Pitts in [6] introduce a system, called FM, with a special quantifier for expressing freshness of names. The intuitive meaning of $\mathsf{N}y.p$ is "p holds for y a fresh name". N resembles both $\forall$ and $\exists$, and it satisfies the rules:

$$\frac{\Gamma, y\#\boldsymbol{x} \vdash p}{\Gamma \vdash \mathsf{N}y.p} \qquad \frac{\Gamma \vdash \mathsf{N}y.p \quad \Gamma, p, y\#\boldsymbol{x} \vdash q}{\Gamma \vdash q}$$

where $\boldsymbol{x}$ is the "support" of p. In the Theory of Contexts, $\mathsf{N}y.p$ and $y\#\boldsymbol{x}$ can be encoded as follows:

$$\mathsf{N}y.p \triangleq \forall y{:}\upsilon.y \not\in^{\upsilon\to o} (\lambda y{:}\upsilon.p) \Rightarrow p \qquad y\#\boldsymbol{x} \triangleq y \not\in^o p$$

Rules, corresponding to the ones above, can then be easily derived using the Theory of Contexts. The abstraction ($x.a$) and instantiation ($a@x$) operators are taken as primitives in FM. In our approach both can be rendered naturally, using the features of the metalanguage: the first as λ-abstraction, the latter as application. Notice that instantiation in FM is only partially defined, *i.e.*, when x is not in the support of a, *i.e.*, the free variables of a. The *fresh* operator, on the other hand, cannot be encoded at the level of terms. Its uses however can be recovered at the level of predicates, see the paragraph below. It is interesting to notice that this condition has a bearing on the fact that AC! holds in FM.

Correspondingly, suitable adaptations of our Theory of Contexts are validated in the FM. More experiments need to be carried out to verify the adequacy of these translations, and to compare the advantages of using FM versus the Theory of Contexts in, possibly mechanized, proof search.

Programming in Υ. Currently there is a great deal of research on programming languages featuring contexts and binding structures as datatypes [18, 13, 5]. The term language of Υ could be extended naturally to a functional programming language to this end, possibly adding a fixed point operator. However, in view of Remark 4 this would not be a very expressive language. A much better alternative would be to define separately a programming language whose semantics

would be the functional relations in Υ. An operational semantics for this language could be given directly or in a logic programming style. A program logic for this language should be derivable from Υ. More work needs to be done in this direction.

Completeness of the Theory of Contexts. An open question.

References

1. A. Bucalo, M. Hofmann, F. Honsell, M. Miculan, and I. Scagnetto. Using functor categories to explain and justify an axiomatization of variables and schemata in HOAS. In preparation, 2001.
2. J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order syntax in Coq. In *Proc. of TLCA'95*, volume 905 of *Lecture Notes in Computer Science*, Edinburgh, Apr. 1995. Springer-Verlag. Also appears as INRIA research report RR-2556, April 1995.
3. J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher order abstract syntax. Technical Report CMU-CS-96-172, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, Aug. 1996.
4. M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In Longo [12], pages 193–202.
5. M. J. Gabbay. *A Theory of Inductive Definitions With α-equivalence*. PhD thesis, Trinity College, Cambridge University, 2000.
6. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In Longo [12], pages 214–224.
7. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.
8. M. Hofmann. Semantical analysis of higher-order abstract syntax. In Longo [12], pages 204–213.
9. F. Honsell and M. Miculan. A natural deduction approach to dynamic logics. In S. Berardi and M. Coppo, editors, *Proc. of TYPES'95*, number 1158 in Lecture Notes in Computer Science, pages 165–182, Turin, Mar. 1995. Springer-Verlag, 1996. A preliminar version has been communicated to the *TYPES'94 Annual Workshop*, Båstad, July 1994.
10. F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (co)inductive type theory. *Theoretical Computer Science*, 253(2):239–285, 2001. First appeared as a talk at TYPES'98 annual workshop.
11. INRIA. *The Coq Proof Assistant*, 2000.
12. G. Longo, editor. *Proceedings, Fourteenth Annual IEEE Symposium on Logic in Computer Science*, Trento, Italy, 2–5 July 1999. The Institute of Electrical and Electronics Engineers, Inc., IEEE Computer Society Press.
13. R. McDowell and D. Miller. A logic for reasoning with higher-order abstract syntax. In *Proc. 12^{th} LICS*. IEEE, 1997.
14. M. Miculan. *Encoding Logical Theories of Programs*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, Mar. 1997.
15. M. Miculan. Encoding and metareasoning of call-by-name λ-calculus. Available at `http://www.dimi.uniud.it/~miculan/CoqCode/HOAS`, 2000.
16. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Inform. and Comput.*, 100(1):1–77, 1992.

17. F. Pfenning and C. Elliott. Higher-order abstract syntax. In *Proc. of ACM SIGPLAN '88 Symposium on Language Design and Implementation*, pages 199–208, Atlanta, Georgia, June 1988. The Association for Computing Machinery.
18. A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction. 5th International Conference, MPC2000, Ponte de Lima, Portugal, July 2000. Proceedings*, volume 1837 of *Lecture Notes in Computer Science*, pages 230–255. Springer-Verlag, Heidelberg, 2000.
19. C. Röckl, D. Hirschkoff, and S. Berghofer. Higher-order abstract syntax with induction in Isabelle/HOL: Formalising the π-calculus and mechanizing the theory of contexts. In F. Honsell and M. Miculan, editors, *Proc. FOSSACS 2001*, volume 2030 of *LNCS*, pages 359–373. Springer-Verlag, 2001.
20. I. Scagnetto. *Reasoning on Names In Higher-Order Abstract Syntax*. PhD thesis, Dipartimento di Matematica e Informatica, Università di Udine, 2002. In preparation.