Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 104 (2004) 99–112
www.elsevier.com/locate/entcs

Plug and Play the Theory of Contexts in Higher-Order Abstract Syntax

Alberto Ciaffaglione and Ivan Scagnetto

Dipartimento di Matematica e Informatica, Università di Udine, Italy
[ciaffagl,scagnett]@dimi.uniud.it

Abstract

We illustrate the pragmatic aspects of the Theory of Contexts, recently proposed as a general approach for reasoning on languages with binders in Higher-Order Abstract Syntax, through two working examples: λ-calculus and Abadi and Cardelli's impς-calculus.

Keywords: Logical Frameworks, Interactive Theorem Proving, Binding Operators, Higher-Order Abstract Syntax, Theory of Contexts.

1 Introduction

In recent years there has been a growing interest in using higher-order type theory-based Logical Frameworks (LFs) for defining and reasoning about languages with binders. A very promising line of approach is the Higher-Order Abstract Syntax (HOAS) technique [6,4,10], which allows to delegate to metalanguages the machinery for dealing with α-conversion and capture-avoiding substitution of terms for variables. This feature is a relevant advantage in respect to first-order techniques, like de Bruijn indexes or explicit names, because encoding and managing α-conversion and substitutions is a non trivial task from the point of view of computer aided formal reasoning.

In the HOAS approach, binders are represented by means of *higher-order constructors*, hence α-equivalence and substitution are provided by the LF itself. However, it is well known that HOAS presents some drawbacks. First of all, object level variables cannot be encoded by means of metalanguage variables using induction, because this choice would introduce exotic terms

[4]. Next, it is difficult to reason by induction and to use recursion over contexts, because they are rendered as functional terms. Finally, the major virtue of HOAS bites back, in the sense that one looses the possibility of proving properties over the mechanisms delegated to the metalanguage.

One approach among the ones proposed to overcome these problems is the *Theory of Contexts* [7], which arises as a general methodology for reasoning about object systems in HOAS, based on an *axiomatic* standpoint. The gist is to extend the working framework with a set of axioms, in order to capture some basic and natural properties of *names* and *contexts*. The main advantage of this technique is that it requires a very low mathematical and logical overhead: it does not actually require to introduce new abstraction and concretion operators, allowing to model abstraction with λ-abstraction and instanciation with functional application. Hence, the Theory of Contexts can be easily used in existing proof environments without the need of any redesign of the system. This fact is indeed the spot of the present work: the authors try to motivate the reader using promptly the axiomatic framework, along the vein of the very recent successful experiments with different language paradigms in both typed and untyped settings [14,2,3].

The structure of the document, which provides users with a light and fast consultation reference, is as follows. In Section 2 we give a deeper account about the HOAS technique, then in Section 3 we present briefly and informally the Theory of Contexts, which is used explicitly for managing the working examples of Section 4: untyped λ-calculus and Abadi and Cardelli's impς-calculus. Finally, in Section 5 we report an interesting result, namely, the derivation of higher-order induction from first-order one together with the axioms of the Theory of Contexts. This should help to understand the interaction between all the axioms and their expressivity.

2 Reconciling HOAS with induction

It is well-known that in a higher-order type theory without native support for inductive types, like for example the Edinburgh LF [6], the natural choice for representing a binder (*e.g.* λ) is a *full* HOAS constructor of type $(\Lambda \rightarrow \Lambda) \rightarrow \Lambda$. This allows to delegate not only α-conversion, but also substitution of terms for variables to the metalanguage, freeing the user from the burden of encoding them. However, in order to take advantage of the inductive features of a metalanguage like the *Calculus of Inductive Constructions* (CIC), it is not possible to resort to a full HOAS approach, because constructor types like $(\Lambda \rightarrow \Lambda) \rightarrow \Lambda$ violate the positivity condition of inductive constructors. Hence it is necessary to introduce a separate type for variables, say Var, with the consequence that capture-avoiding substitution of terms for variables is no more delegated to the metalevel (*weak* `HOAS`). Therefore binders are encoded by constructors with negative occurrences of the type representing variables, *i.e.* $(Var \to \Lambda) \to \Lambda$. Correspondingly, Var has not to be defined as an inductive type, because this would introduce *exotic terms* in the framework: exotic terms are terms not corresponding to any expression of the object language, which have to be ruled out by extra "well-formedness" judgments in order to obtain an adequate, faithful encoding [4].

3 The Theory of Contexts

In order to retain the advantages deriving from the use of inductive features of `LF`s and their implementations, it is convenient to adopt a *weak* `HOAS` approach for encoding object languages, as explained in Section 2. This means that we introduce a separate type representing names/variables; moreover, this type is not an inductive one, in order to avoid exotic terms. For instance, we declare in CIC a type which is neither defined effectively, nor proved as a judgment:

```
Parameter var: Set.
```

Then, the object language at hand can be encoded by means of an inductive type `term`, where binders are represented by constants of functional type `(var -> term) -> term`. Thus, since bound variables are rendered by metavariables of type `var`, we gain α-conversion for free.

However, as pointed out in Section 1, the current implementations of most `LF`s do not provide an adequate support for higher-order encodings. When formally developing metatheoretical results about the encoded language, the following common problems actually arise:

(i) one cannot access the notions related to the mechanisms delegated to the metalanguage (*e.g.*, during a proof we cannot access the information concerning the freshness of a bound variable);

(ii) the system does not provide any induction principles over higher-order terms.

There are several attempts in the literature for regaining some expressivity about `HOAS`-based encodings (see, *e.g.*, [4,13,5]). The approach we describe in this paper is known as the *Theory of Contexts* [7]: it consists of a set of axiom schemata about some basic properties of names/variables and contexts (*i.e.*, terms with "holes") over them. The main advantage of an axiomatic approach is that it can be easily plugged in existing `LF`s (provided they support the introduction of new axioms) without requiring any redesign of the system, nor great encoding efforts.

Axioms of the Theory of Contexts.

We introduce the axioms in an informal way, in order to allow the reader grasping the main underlying ideas; and from now on we will use the terminology "variables" to intend names/variables.

The first axiom requires that there always exists a fresh, *i.e.* free, variable with respect to a term, thus capturing the intuition that, since terms are finite entities, they cannot contain all the variables. The same axiom can be stated *w.r.t.* lists of variables instead of terms, since we can always obtain from a term the list of its free variables:

$$\text{unsaturation: } \forall M.\ \exists x.\ x \notin \mathsf{FV}(M) \qquad \text{unsaturation': } \forall L.\ \exists x.\ x \notin L$$

The second axiom requires the decidability of equality over variables:

$$LEM_{eq}:\ \forall x, y.\ x = y \vee x \neq y$$

In a classical framework, this axiom is just an instance of the Law of Excluded Middle; on the other hand, it represents the minimal classical flavour we need in an intuitionistic setting.

The core of the Theory of Contexts is represented by the β-expansion and extensionality axioms. These allow to carry out proofs by reasoning over the structure of higher-order terms; indeed they are *schemata* of axioms, since can be formulated for contexts with n holes. For instance, when instanciated to unary contexts (*i.e.*, terms with one hole), they appear as follows:

$$\begin{aligned} \beta\text{-expansion: } & \forall M, \forall x.\ \exists N[\cdot].\ x \notin \mathsf{FV}(N[\cdot]) \wedge M = N[x] \\ \text{extensionality: } & \forall M[\cdot], \forall N[\cdot], \forall x.\ x \notin \mathsf{FV}(M[\cdot]) \cup \mathsf{FV}(N[\cdot]) \\ & \wedge (M[x] = N[x]) \Rightarrow M[\cdot] = N[\cdot] \end{aligned}$$

Essentially, β-expansion allows to generate a new context N with one new hole from another context M, by abstracting over a variable. Extensionality allows to infer that two contexts are equal, by testing the equivalence of their application to a fresh variable. Due to lack of space, we cannot give the details about the soundness and the expressivity of this theory [14]. However, for what concerns the expressive power, it should suffice saying that the axioms of the Theory of Contexts, together with the complete induction principle on natural numbers, allow to derive a higher-order structural induction principle for the object language syntax (§4.5.2 of [14]).

4 The Theory of Contexts at work

In this section we briefly illustrate two effective encoding examples, showing how to work in any sufficiently powerful `LF` with the axioms described in

Section 3. Before going into the details, it is worth distinguishing between two kinds of object languages, namely untyped and typed ones. In the latter case, the unsaturation axiom has to be slightly modified:

$$\text{unsaturation-typed: } \forall M, \forall \boldsymbol{T}.\ \mathcal{R}(\boldsymbol{T}) \Rightarrow \exists x.\ x \notin \mathsf{FV}(M) \wedge \mathcal{Q}(x, \boldsymbol{T})$$

where $\mathcal{Q}(x, \boldsymbol{T})$ represents the properties which we assume about the new "typed" variable x. Indeed, in a typed object language, variables are not simple placeholders, but they carry some information with them (*e.g.*, their type) which is represented by the predicate $\mathcal{Q}$. The role of predicate $\mathcal{R}$ is more subtle and we can suggest its meaning by means of an example: since in typed object languages variables are typed as well, the simplest instanciation of this statement establishes that there are infinite variables for every given type, which corresponds to take $\mathcal{R}(A) \triangleq \mathit{true}$ and $\mathcal{Q}(x, A) \triangleq x{:}A$. However, it may be the case that in some object languages there are types, such that we cannot freely assume the existence of "fresh" variables over them. In other words, the simplest form of the unsaturation axiom would be inconsistent, since it always allows to infer the existence of a new typed variable for every type. Hence, before applying the unsaturation axiom, we must verify that the given type satisfies some constraints (represented by the predicate $\mathcal{R}$) which avoid the introduction of inconsistencies. For instance, in the case of the encoding of $\mathsf{imp}\varsigma$ (see the example below), $\mathcal{R}$ states that T must be inhabited.

It is also worth noticing that, before instanciating the axioms, it is necessary to encode the *non-occurrence* predicate "$\not\in$". As we will see below, this obligation is completely trivial, since it is syntax-driven [7].

We proceed illustrating the two real case studies of untyped λ-calculus and typed $\mathsf{imp}\varsigma$-calculus. For our encodings, we use from now on the proof assistant Coq as LF; clearly, the presentation can be easily grasped by users of alternative proof environments and type theories.

The untyped λ-calculus.

The syntax of the object language we focus on is well known:

$$\Lambda:\ M, N ::= x \mid MN \mid \lambda x.M$$

Hence we have the following specification in Coq:

```
Parameter var: Set.
Inductive tm : Set:= is_var : var -> tm
                   | app    : tm -> tm -> tm
                   | lam    : (var -> tm) -> tm.
```

Notice the higher-order type of the `lam` constructor, which allows to represent the object language variables by means of Coq's metavariables of type `var`,

$$
\begin{array}{lll@{\qquad}lll}
\epsilon^{\Lambda}_{X} & : & \Lambda_X \longrightarrow \mathtt{tm}_X & \delta^{\Lambda}_{X} & : & \mathtt{tm}_X \longrightarrow \Lambda_X \\
\epsilon^{\Lambda}_{X}(x) & \triangleq & (\mathtt{is\_var}\ \mathtt{x}) & \delta^{\Lambda}_{X}((\mathtt{is\_var}\ \mathtt{x})) & \triangleq & x \\
\epsilon^{\Lambda}_{X}(MN) & \triangleq & (\mathtt{app}\ \epsilon^{\Lambda}_{X}(M)\ \epsilon^{\Lambda}_{X}(N)) & \delta^{\Lambda}_{X}((\mathtt{app}\ \mathtt{M}\ \mathtt{N})) & \triangleq & \delta^{\Lambda}_{X}(M)\delta^{\Lambda}_{X}(N) \\
\epsilon^{\Lambda}_{X}(\lambda x.M) & \triangleq & (\mathtt{lam}\ [\mathtt{x} : \mathtt{var}]\epsilon^{\Lambda}_{X\cup\{x\}}(M)) & \delta^{\Lambda}_{X}((\mathtt{lam}\ [\mathtt{x} : \mathtt{var}]\mathtt{M})) & \triangleq & \lambda x.\delta^{\Lambda}_{X\cup\{x\}}(\mathtt{M})
\end{array}
$$

Fig. 1. Encoding and decoding functions for the untyped λ-calculus

and to delegate the α-conversion mechanism to the metalanguage. In the following, for a finite set of variables $X \triangleq \{x_1, \ldots, x_n\}$, we will denote by Γ_X the typing environment $\{\mathtt{x1} : \mathtt{var}, \ldots, \mathtt{xn} : \mathtt{var}\}$. The representation is formally specified by means of the encoding and decoding functions of Figure 1, where Λ_X denotes the λ-terms with free variables in X and $\mathtt{tm}_X$ the terms of type `tm` derivable from the environment Γ_X. Hence we have the following adequacy result:

Proposition 4.1 *For each finite set of variables X, $\varepsilon^{\Lambda}_{X}$ is a compositional bijection between Λ_X and $\mathtt{tm}_X$.*

As previously remarked, the encoding of the non-occurrence predicate $\notin$ is syntax-driven (one rule for each constructor of type `tm`):

```
Inductive notin [x:var]: tm -> Prop :=
    notin_var: (y:var) ~x=y -> (notin x (is_var y))
  | notin_app: (M,N:tm) (notin x M) -> (notin x N) ->
                        (notin x (app M N))
  | notin_lam: (y:var) (M:var->tm) ((y:var)(notin x (M y))) ->
                       (notin x (lam M)).
```

Now we can introduce the instanciation of the Theory of Contexts for the case at hand:

```
Axiom dec_var  : (x,y:var) x=y \/ ~x=y.
Axiom unsat    : (M:tm) (Ex [x:var] (notin x M)).
Axiom exp      : (M:tm) (x:var)
                 (Ex [N:var->tm] (notin x (lam N)) /\ M=(N x)).
Axiom ext      : (M,N:var->tm) (x:var)
                 (notin x (lam M)) -> (notin x (lam N)) ->
                 (M x)=(N x) -> M=N.
```

As outlined in Section 3, β-expansion and extensionality are *schemata* of axioms, hence they can be stated for contexts with an arbitrary number of "holes". For example, we could declare the variants:

```
Axiom ho_exp : (M:var->tm)(x:var)(Ex [N:var->var->tm]
               (notin x (lam [y:var](lam (N y)))) /\ M=(N x)).
Axiom ho_ext : (M,N:var->var->tm)(x:var)
```

```
(notin x (lam [y:var](lam (M y)))) ->
(notin x (lam [y:var](lam (N y)))) ->
(M x)=(N x) -> M=N.
```

According to our experience, it is never needed to go beyond the variants involving contexts with more than three holes. Working with this simple theory it is possible to prove some involved metatheoretic results; for instance in [11], after adding the encoding of a type system, the formal proof of Subject Reduction is carried out for a call-by-name λ-calculus.

Abadi and Cardelli's impς-calculus.

The impς-calculus [1] is an imperative object-based calculus featuring objects, dynamic lookup, method update, cloning and local declarations:

$Term: \ a, b ::=$	x	variable
	$[l_i = \varsigma(x_i)b_i]^{i \in I}$	object
	$a.l$	method invocation
	$a.l \leftarrow \varsigma(x)b$	method update
	$clone(a)$	cloning
	$let\ x = a\ in\ b$	local declaration

Notice that ς binds x_i, x in b_i, b, and *let* binds x in b, so the syntax can be encoded in Coq as follows:

```
Parameter var : Set. Definition lab := nat.
Inductive term : Set := isvar: var -> term
                | obj   : (list (lab * (var -> term))) -> term
                | clone: term -> term
                | call  : term -> lab -> term
                | over  : term -> lab -> (var -> term) -> term
                | let   : term -> (var -> term) -> term.
```

In the present case, the higher-order constructors are `obj` and `over`, binding the host object in method bodies, and `let`. Note that natural numbers play the role of labels, *i.e.* names of methods. The adequacy of the encoding can be stated and proved similarly to the case of λ-calculus.

The impς-calculus features a first-order type system *à la* Curry with subtyping: the only type constructor is that for object types, *i.e.* $TType: \ A, B ::= [l_i : A_i]^{i \in I}$. Once we have encoded also types (`ttype`) and the non-occurrence predicate (`fresh`), we adopt the Theory of Contexts for proving the rather involved property of Subject Reduction. This assures that the dynamic semantics is coherent with the static semantics, so it requires to formalize two reduction and typing judgments for impς. Therefore we need variables for dealing both with values (in reductions) and types (in typings), thus we state the unsaturation axiom in two flavours, so splitting the universe `var`. The first axiom corresponds to the case of using metavariables as *placeholders*, and is useful in conjunction with typing properties. The second axiom reflects the use of metavariables for *variables* of the object language, and relates results and (their) types, provided they are proved to be consistent through `type_val`:

```
Axiom unsat_type: (A:ttype)(xl:(list var))
      (EX x | (fresh x xl) /\ (typenv x)=A).
Axiom unsat_val: (v:val)(A:ttype)(type_val v A)->(xl:(list var))
      (EX x | (fresh x xl) /\ (stack x)=v /\ (typenv x)=A).
```

Notice that the functions `stack` and `typenv` implement derivations contexts for reduction and typing, respectively. The above axioms and the LEM_{eq} one, which is stated identically to the case of λ-calculus, are very natural and useful for dealing with hypothetical assertions, which are typical of *natural deduction* style of proof. Consider *e.g.* the typing rule for the *let* construct:

$$\frac{\Delta \vdash a : A \quad \Delta, x{:}A \vdash b : B}{\Delta \vdash let\ x = a\ in\ b : B}\ (Type{-}Let)$$

This rule is rendered in `Coq` using an hypothetical premise for representing assumptions which have to be discharged in the conclusion (the function `typenv` implements the typing context Δ):

```
t_let: (a:term) (b:var->term) (A,B:ttype)
       (type a A) -> ((x:var)(typenv x)=(A)->(type (b x) B)) ->
       (type (let a b) B)
```

Very often it is required to derive the conclusions of hypothetical assertions, like, *e.g.*, `(type (b x) B)`: in such a case, the unsaturation and LEM_{eq} allow to introduce a new, fresh variable `x` in the proof derivation context Δ, thus allowing to use the premise `(typenv x)=(A)`.

Pragmatic remarks.

Since the Theory of Contexts has been first introduced in order to carry out formal proofs about the metatheory of the encoded systems, it is interesting to highlight the general application pattern of the axioms. During our formal developments, we always find a common problem: to guarantee that some crucial property (*e.g.*, strong late bisimilarity for the π-calculus, capture-avoiding substitution and reduction relation for the untyped λ-calculus, typing of higher-order terms for the impς-calculus) is preserved by *fresh renamings*, *i.e.*, by replacing a given variable with a fresh one. All these lemmata are instances of the following pattern:

$$\frac{\text{for some } x \notin \bigcup_{i=1}^{n} \mathsf{FV}(C_i[\cdot]) : \mathcal{R}(C_1[x], \ldots, C_n[x])}{\text{for all } y \notin \bigcup_{i=1}^{n} \mathsf{FV}(C_i[\cdot]) : \mathcal{R}(C_1[y], \ldots, C_n[y])} \tag{1}$$

where $\mathcal{R}$ is a given n-ary relation and $C_1[\cdot] \ldots, C_n[\cdot]$ are variables ranging over contexts of given syntactic categories. For example, in the case of the impς-calculus, one renaming looks like:

```
Lemma rename: (m:var->term) (A:TType) (x,y:Var)
              (type (m x) A) -> (typenv x) = (typenv y) ->
              (type (m y) A).
```

Usually, this kind of properties is proved "with pencil and paper" by carrying out a structural induction either on the derivation of the premise $\mathcal{R}(C_1[x], \ldots, C_n[x])$, or on one of the arguments $C_i[x]$ $(1 \le i \le n)$, or else on a "measure" of an argument (*e.g.*, the number of symbols it contains). However, since Coq tactics deal not adequately with higher-order unification, we are forced to prove a preliminary version of the renaming lemma introducing by hand the necessary unifications: this allows to recover sufficient information on the structure of the contexts $C_i[\cdot]$ from their instanciations $C_i[x]$. In other words, we "lift" structural information to the level of functional terms, using the β-expansion and the extensionality axioms. Such a lifting follows a general pattern. First we replace the original goal with the following one:

$$\frac{\text{for some } x \notin \bigcup_{i=1}^{n} \mathsf{FV}(C_i[\cdot]), T_1 = C_1[x], \ldots, T_n = C_n[x] : \mathcal{R}(T_1, \ldots, T_n)}{\text{for all } y \notin \bigcup_{i=1}^{n} \mathsf{FV}(C_i[\cdot]) : \mathcal{R}(C_1[y], \ldots, C_n[y])} \tag{2}$$

where $T_1, \ldots, T_n$ are plain terms and $T_1 = C_1[x], \ldots, T_n = C_n[x]$ are the necessary unifications. Clearly we can infer (1) from (2) by taking $T_i = C_i[x]$.

During the proof of (2), the inductive hypothesis gives us some structural information on $T_1, \ldots, T_n$. Then, using the β-expansion axiom, we can expand the latter into contexts applied to x, yielding the equations $T_1 = T'_1[x], \ldots, T_n = T'_n[x]$, where $x \notin \bigcup_{i=1}^{n} \mathsf{FV}(T'_i[\cdot])$ (differently from $C_i[\cdot]$, $T'_i[\cdot]$ is not a variable, but a concrete context). By transitivity, we obtain the equations $C_i[x] = T'_i[x]$; thus, by extensionality, we get $C_i[\cdot] = T'_i[\cdot]$, *i.e.*, the structural information we needed on the variable $C_i[\cdot]$. Such an information can then be transferred to the instanciations over y in the current goal, in order to apply the suitable constructor of $\mathcal{R}$ and solve the subsequent subgoal by means of the inductive hypothesis.

Now, it should be clear why we choose Leibniz equality in our axioms, instead of an external equality predicate. Indeed, the former represent Coq's

$\beta\delta\iota$-conversion, whence equalities involving it can be used to rewrite terms into equivalent ones by means of the `Rewrite` tactic.

5 Higher-order induction

The soundness of the Theory of Contexts is proved in [14]. However, as far the completeness is concerned, we do not have yet a result stating the expressive power of the axioms *w.r.t.* some known logic system. In this section we prove an important result in this direction, namely, the derivability of the higher-order induction principle by means of the complete induction principle on natural numbers and the axioms of the Theory of Contexts. In order to spell out all the details, we will consider again the encoding of untyped λ-calculus in `Coq`. The complete source code of the proof is available in [14].

5.1 Encoding of syntax

Let us consider the encoding of untyped λ-calculus (see the first case study of Section 4); we introduce the following *measure relation* `l`:

```
Inductive l: tm -> nat -> Prop:=
  l_var : (x:var)(l (is_var x) (S O))
| l_app : (M,N:tm)(n1,n2:nat)(l M n1) -> (l N n2) ->
          (l (app M N) (S (plus n1 n2)))
| l_lam : (M:var->tm)(n:nat)((y:var)(l (M y) n)) ->
          (l (lam M) (S n)).
```

Intuitively `(l M n)` holds if and only if `M` contains exactly `n` occurrences of constructors belonging to the type `tm`. As we will see in the next subsection, this definition will be crucial during the proof development in `Coq`.

5.2 The formal development

The first results we need concern properties of the measure relation `l`; first, we show that `l` is preserved by fresh renaming:

```
Lemma L_RW: (n:nat)(M:tm)(l M n) ->
            (x:var)(N:var->tm)(notin x (lam N)) ->
            M=(N x) -> (y:var)(l (N y) n).
```

The proof technique used is a complete induction on `n`. We notice that complete induction on natural numbers is trivially derivable from the induction principle `nat_ind`, which is automatically provided by `Coq` on type `nat`. The reason for using complete induction is that it allows to apply the inductive hypothesis to any term structurally smaller than that of the current hypothesis, not only to the immediate subterm of the latter (which is, instead, the only possibility offered by the induction principle `tm_ind` provided by `Coq`). Hence, we can "mimick" a complete induction principle on the structure of terms by means of a complete induction on the number of constructors' occurrences of terms. This is fundamental in proving renaming results like `L_RW`, since, in the cases involving binders, there is the need to apply the inductive hypothesis two times before concluding. The first application is carried out only to replace all the occurrences of the generic variable introduced by the `l_lam` rule. Indeed, being generic, such a variable is not generally fresh, and this fact is in conflict with the `notin` judgment appearing in the inductive hypothesis. A glance at the relative `Coq` session will make the argument clear:

```
...
n0 : nat
y : var
x0 : var->var->tm
...
============================
(l (lam (x0 y)) (S n1))
```

Here we are considering the case relative to the binder `lam`; hence, we must apply rule `l_lam` (`Apply l_lam; Intro.`), thus getting the following proof environment:

```
n : nat
n0 : nat
H : (m:nat)(lt m n0)->(M:tm)
      (l M m)->(x:var; N:(var->tm))
        (notin x (lam N))->M=(N x)->(y:var)(l (N y) m)
x : var
N : var->tm
H1 : (notin x (lam N))
y : var
M0 : var->tm
n1 : nat
H5 : (S n1)=n0
H3 : (y:var)(l (M0 y) n1)
x0 : var->var->tm
H7 : (notin x (lam [_:var](lam (x0 _))))
H8 : M0=(x0 x)
H2 : (lam (x0 x))=(N x)
```

```
H6 : N=([_:var](lam (x0 _)))
y0 : var
============================
(l (x0 y y0) n1)
```

Naïvely applying the inductive hypothesis `H` for replacing `y` with `x` does not work, since, among the new subgoals, we have to prove `(notin x (lam [_:var](x0 y0)))` and this is not possible, because `y0`, being generic, could be equal to `x`. The right approach consists of replacing `y0` with a new fresh variable (obtained by means of `unsat`) and then replacing `y` with `x`. These operations amount to applying two times the inductive hypothesis.

Once `L_RW` is derived, we can prove the totality of `l` *w.r.t.* the first argument by means of a structural induction on it:

```
Lemma L_TOT: (M:tm)(Ex [n:nat](l M n)).
```

Now, we have all the results we need in order to derive, again by a complete induction on n, the following lemma (notice the generic variable of the schematic judgment `((y:var)(P [x:var](M x y)))`):

```
Lemma PRE_HO_TM_IND: (P:(var->tm)->Prop)
                     ((x:var)(P [_:var](is_var x))) ->
                     (P is_var) ->
                     ((M,N:var->tm)(P M) -> (P N) ->
                      (P [x:var](app (M x) (N x)))) ->
                     ((M:var->var->tm)
                      ((y:var)(P [x:var](M x y))) ->
                      (P [x:var](lam (M x)))) ->
                     (n:nat)(M:tm)(l M n) ->
                     (N:var->tm)(x:var)(notin x (lam N)) ->
                     (N x)=M -> (P N).
```

The main result, *i.e.* the higher-order induction principle for terms of type `var->tm` can be obtained as a straightforward corollary of `PRE_HO_TM_IND`:

```
Lemma HO_TM_IND: (P:(var->tm)->Prop)
                 ((x:var)(P [_:var](is_var x))) ->
                 (P is_var) ->
                 ((M,N:var->tm)(P M) -> (P N) ->
                  (P [x:var](app (M x) (N x)))) ->
                 ((M:var->var->tm)
                  ((y:var)(P [x:var](M x y))) ->
                  (P [x:var](lam (M x)))) ->
                 (M:var->tm)(P M).
```

The axioms of β-expansion and extensionality play a fundamental role in proving lemmata `L_RW` and `PRE_HO_TM_IND`, used for "transferring" structural information from terms of type `tm` to contexts of type `var->tm`. This fact is explained in more detail in the pragmatic remarks at the end of Section 4. The whole approach can be adapted (changing the definition of the measure relation `l`) for deriving higher-order induction principles for terms of type `var->var->tm`, `var->var->var->tm` and so on. For instance, the measure relation for unary contexts of type `var->tm` is the following:

```
Inductive ho_l : (var->tm)->nat->Prop :=
  ho_l_var1 : (ho_l [_:var](is_var _) (S O))
| ho_l_var2 : (x:var)(ho_l [_:var](is_var x) (S O))
| ho_l_app : (M,N:var->tm; n1,n2:nat)(ho_l M n1)->(ho_l N n2)->
             (ho_l [_:var](app (M _) (N _)) (S (plus n1 n2)))
| ho_l_lam : (M:var->var->tm)(n:nat)
             ((y:var)(ho_l [_:var](M _ y) n)) ->
             (ho_l [_:var](lam (M _)) (S n)).
```

6 Conclusion

The Theory of Contexts is an axiomatic attempt to capture some very primitive properties about names and contexts in weak HOAS. Its first two axioms (unsaturation and decidability of the equality over names) describe two fundamental and well-accepted assumptions about the practical use of names. The role of the remaining axiom schemata is instead more subtle, since they reveal their expressive power when used together. Indeed, extensionality allows to infer the needed structural information about an "abstract" context M from a "concrete" one N', obtained by a β-expansion of a ground term N in $N'[x]$ (provided we can prove that $M[x] = N'[x]$ for $x \notin \mathsf{FV}(M) \cup \mathsf{FV}(N)$). This is the main idea behind the derivation of the higher-order induction principle, addressed in Section 5.

The Theory of Contexts has been used in combination with weak HOAS for carrying out many case studies in recent years (see, *e.g.*, [2,3,8,11,9,12]). These works deal with very different paradigms, and produce machine-checked error-prone proofs of very involved metatheoretical properties in the different object systems. However, all these works seem to adopt an ad-hoc variant of the axioms without following a common methodology. This paper tries to disclaim this wrong impression, and aims to give a brief roadmap to follow systematically in order to instanciate the axiom schemata and to work with a weak HOAS-based encoding.

References

[1] M. Abadi and L. Cardelli. A theory of objects. Springer-Verlag, 1996.

[2] A. Ciaffaglione, L. Liquori, and M. Miculan. Imperative Object-based Calculi in (Co)Inductive Type Theories. In *Proc. of LPAR*, *Lecture Notes in Artificial Intelligence 2850*, 2003.

[3] A. Ciaffaglione, L. Liquori, and M. Miculan. Reasoning on an Imperative Object-based Calculus in Higher-Order Abstract Syntax. In *Proc. of MERLIN*, *ACM*, 2003.

[4] J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order syntax in Coq. In *Proc. of TLCA*, *Lecture Notes in Computer Science 905*, 1995.

[5] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing ?*, to appear.

[6] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of ACM 40(1)*, 1993.

[7] F. Honsell, M. Miculan, and I. Scagnetto. An axiomatic approach to metareasoning on systems in higher-order abstract syntax. In *Proc. of ICALP*, *Lecture Notes in Computer Science 2076*, 2001.

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

[9] F. Honsell, M. Miculan, and I. Scagnetto. The theory of contexts for first-order and higher-order abstract syntax. In *Proc. of TOSCA*, *Electronic Notes in Theoretical Computer Science 62*, 2001.

[10] M. Miculan. Encoding logical theories of programs. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, 1997.

[11] M. Miculan. Developing (meta)theory of lambda-calculus in the theory of contexts. In *Proc. of MERLIN*, *Electronic Notes in Theoretical Computer Science 58.1*, 2001.

[12] M. Miculan and I. Scagnetto. Ambient calculus and its logic in the calculus of inductive constructions. In *Proc. of LFM*, *Electronic Notes in Theoretical Computer Science 70.2*, 2002.

[13] F. Pfenning and C. Schürmann. System description: Twelf, a meta-logical framework for deductive systems. In *Proc. of CADE*, *Lecture Notes in Artificial Intelligence 1632*, 1999.

[14] I. Scagnetto. Reasoning about Names in Higher-Order Abstract Syntax. Cs 2002/4, Dipartimento di Matematica e Informatica, Università di Udine, 2002.