Electronic Notes in Theoretical Computer Science 104 (2004) 99–112 

[www.elsevier.com/locate/entcs](http://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, Universit`a 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 imp*ς*-calculus.

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

# 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 lan- guages with binders. A very promising line of approach is the Higher-Order Abstract Syntax (HOAS) technique [[6](#_bookmark13),[4](#_bookmark8),[10](#_bookmark17)], which allows to delegate to meta- languages 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, be- cause 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

1571-0661 © 2004 Published by Elsevier B.V.Open access under [CC BY-NC-ND license.](http://creativecommons.org/licenses/by-nc-nd/3.0/)

doi:10.1016/j.entcs.2004.09.022

[[4](#_bookmark8)]. 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](#_bookmark14)], 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 over- head: 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](#_bookmark21),[2](#_bookmark9),[3](#_bookmark10)].

The structure of the document, which provides users with a light and fast consultation reference, is as follows. In Section [2](#_bookmark1) we give a deeper account about the HOAS technique, then in Section [3](#_bookmark2) we present briefly and informally the Theory of Contexts, which is used explicitly for managing the working

examples of Section [4](#_bookmark3): untyped *λ*-calculus and Abadi and Cardelli’s imp*ς*-

calculus. Finally, in Section [5](#_bookmark7) we report an interesting result, namely, the derivation of higher-order induction from first-order one together with the ax- ioms of the Theory of Contexts. This should help to understand the interaction between all the axioms and their expressivity.

# 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](#_bookmark13)], the natural choice for representing a binder (*e*.*g*. *λ*) is a *full* HOAS constructor of type (Λ → Λ) → Λ.

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 en- coding 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 (Λ → Λ) → Λ violate the positivity condition of inductive constructors. Hence it is necessary to introduce a separate type for variables, say *V ar*, 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*. (*V ar* → Λ) → Λ. Correspondingly, *V ar* has not to be defined as an induc- tive 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](#_bookmark8)].

# The Theory of Contexts

In order to retain the advantages deriving from the use of inductive features of LFs and their implementations, it is convenient to adopt a *weak* HOAS approach for encoding object languages, as explained in Section [2](#_bookmark1). 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 metavari- ables of type var, we gain *α*-conversion for free.

However, as pointed out in Section [1](#_bookmark0), the current implementations of most LFs 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:

1. 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);
2. 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](#_bookmark8),[13](#_bookmark20),[5](#_bookmark11)]). The approach we describe in this paper is known as the *Theory of Contexts* [[7](#_bookmark14)]: 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 LFs (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 termi- nology “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:

unsaturation: ∀*M.* ∃*x. x* /∈ FV(*M* ) unsaturation’: ∀*L.* ∃*x. x* /∈ *L*

The second axiom requires the decidability of equality over variables:

*LEMeq* : ∀*x, y. x* = *y* ∨ *x* /= *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:

*β*-expansion: ∀*M,* ∀*x.* ∃*N* [·]*. x* /∈ FV(*N* [·]) ∧ *M* = *N* [*x*]

extensionality: ∀*M* [·]*,* ∀*N* [·]*,* ∀*x. x* ∈*/* FV(*M* [·]) ∪ FV(*N* [·])

∧(*M* [*x*] = *N* [*x*]) ⇒ *M* [·] = *N* [·]

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](#_bookmark21)]. 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](#_bookmark21)]).

# 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](#_bookmark2). 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:

unsaturation-typed: 6*M,* 6T *.* Y(T ) ⇒ E*x. x* ∈*/* FV(*M* ) Λ Q(*x,* T )

where Q(*x,* 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 Q. The role of predicate Y 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 Y(*A*) *true* and Q(*x, A*) *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 Y) which avoid

the introduction of inconsistencies. For instance, in the case of the encoding of imp*ς* (see the example below), Y states that *T* must be inhabited.

It is also worth noticing that, before instanciating the axioms, it is neces- sary to encode the *non-occurrence* predicate “/∈”. As we will see below, this obligation is completely trivial, since it is syntax-driven [[7](#_bookmark14)].

We proceed illustrating the two real case studies of untyped *λ*-calculus and typed imp*ς*-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:

Λ : *M, N* ::= *x* | *MN* | *λx.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,

Λ : Λ*X* −→ tm*X*

*G*

*X*

*G*Λ (*x*) (is var x)

*X*

*G*Λ (*MN* ) (app *G*Λ (*M* ) *G*Λ (*N* ))

Λ : tm*X* −→ Λ*X*

*δ*Λ ((is var x)) *x*

*δ*

*X*

*X*

*δ*Λ ((app M N)) *δ*Λ (*M* )*δ*Λ (*N* )

*X X X X X* *X*

*G*Λ (*Zx.M* ) (lam [x : var]*G*Λ

(*M* ))

*δ*Λ ((lam [x : var]M)) *Zx.δ*Λ

(M)

*X X*∪{*x*} *X X*∪{*x*}

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

and to delegate the *α*-conversion mechanism to the metalanguage. In the following, for a finite set of variables *X* {*x*1*,... , xn*}, we will denote by Γ*X* the typing environment {x1 : var*,... ,* xn : var}. The representation is formally specified by means of the encoding and decoding functions of Figure [1](#_bookmark4), where Λ*X* denotes the *λ*-terms with free variables in *X* and 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, ε*Λ

*X*

*is a compositional*

*bijection between* Λ*X and* tm*X.*

As previously remarked, the encoding of the non-occurrence predicate ∈*/*

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](#_bookmark2), *β*-expansion and extensionality are *schemata* of ax- ioms, 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](#_bookmark18)], 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 imp*ς*-calculus.

The imp*ς*-calculus [[1](#_bookmark12)] is an imperative object-based calculus featuring ob- jects, dynamic lookup, method update, cloning and local declarations:

*T erm* : *a, b* ::= *x* variable [*li* = *ς*(*xi*)*bi*]*i*∈*I* object

*a.l* method invocation

*a.l* ← *ς*(*x*)*b* method update

*clone*(*a*) cloning

*let x* = *a in b* local declaration

Notice that *ς* binds *xi,x* in *bi, 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 imp*ς*-calculus features a first-order type system *a` la* Curry with sub- typing: the only type constructor is that for object types, *i*.*e*. *TT ype* : *A, B* ::= [*li* : *Ai*]*i*∈*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 se- mantics is coherent with the static semantics, so it requires to formalize two reduction and typing judgments for imp*ς*. 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 *LEMeq* 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:

∆ ▶ *a* : *A* ∆*, x*:*A* ▶ *b* : *B*

∆ ▶ *let x* = *a in b* : *B*

(*T ype*−*Let*)

This rule is rendered in Coq using an hypothetical premise for representing as- sumptions which have to be discharged in the conclusion (the function typenv implements the typing context ∆):

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 *LEMeq* allow to introduce a new, fresh variable x in the proof derivation context ∆, 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 inter- esting 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 imp*ς*-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:

for some *x* /∈ *n*

*n i*=1

FV(*Ci*[·]) : Y(*C*1[*x*]*,... , Cn*[*x*])

(1)

for all *y* /∈

*i*=1 FV(*Ci*[·]) : Y(*C*1[*y*]*,... , Cn*[*y*])

where Y is a given *n*-ary relation and *C*1[·] *... , Cn*[·] are variables ranging over contexts of given syntactic categories. For example, in the case of the imp*ς*-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 Y(*C*1[*x*]*,*

*... , Cn*[*x*]), or on one of the arguments *Ci*[*x*] (1 ≤ *i* ≤ *n*), or else on a “mea- sure” 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 *Ci*[·] from their instanciations *Ci*[*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:

for some *x* /∈ *n*

*i*=1

FV(*Ci*[·])*, T*1 = *C*1[*x*]*,... , Tn* = *Cn*[*x*] : Y(*T*1*,... , Tn*)

*n*

(2)

for all *y* /∈

*i*=1 FV(*Ci*[·]) : Y(*C*1[*y*]*,... , Cn*[*y*])

where *T*1*,... , Tn* are plain terms and *T*1 = *C*1[*x*]*,..., Tn* = *Cn*[*x*] are the necessary unifications. Clearly we can infer ([1](#_bookmark5)) from ([2](#_bookmark6)) by taking *Ti* = *Ci*[*x*]. During the proof of ([2](#_bookmark6)), the inductive hypothesis gives us some struc- tural information on *T*1*,... , Tn*. Then, using the *β*-expansion axiom, we can

expand the latter into contexts ap plied to *x*, yielding the equations *T*1 =

*T* '[*x*]*,..., Tn* = *T* ' [*x*], where *x* /∈ *n* FV(*T* '[·]) (differently from *Ci*[·], *T* '[·]

1 *n i*=1 *i* *i*

is not a variable, but a concrete context). By transitivity, we obtain the

equations *Ci*[*x*] = *T* '[*x*]; thus, by extensionality, we get *Ci*[·] = *T* '[·], *i*.*e*., the

*i* *i*

structural information we needed on the variable *Ci*[·]. Such an information

can then be transferred to the instanciations over *y* in the current goal, in order to apply the suitable constructor of Y 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

*βδι*-conversion, whence equalities involving it can be used to rewrite terms into equivalent ones by means of the Rewrite tactic.

# Higher-order induction

The soundness of the Theory of Contexts is proved in [[14](#_bookmark21)]. 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](#_bookmark21)].

* 1. *Encoding of syntax*

Let us consider the encoding of untyped *λ*-calculus (see the first case study of Section [4](#_bookmark3)); 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.

* 1. *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 com- plete 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 hypoth- esis, 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’ occur- rences 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)

Na¨ıvely 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 com- plete 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 prov- ing lemmata L\_RW and PRE\_HO\_TM\_IND, used for “transferring” structural in- formation 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](#_bookmark5). 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)).

# Conclusion

The Theory of Contexts is an axiomatic attempt to capture some very prim- itive properties about names and contexts in weak HOAS. Its first two axioms (unsaturation and decidability of the equality over names) describe two funda- mental 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* /∈ FV(*M* ) ∪ FV(*N* )). This is

the main idea behind the derivation of the higher-order induction principle,

addressed in Section [5](#_bookmark7).

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](#_bookmark9),[3](#_bookmark10),[8](#_bookmark15),[11](#_bookmark18),[9](#_bookmark16),[12](#_bookmark19)]). 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.

1. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of ACM* *40(1)*, 1993.
2. 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.
3. F. Honsell, M. Miculan, and I. Scagnetto. *π*-calculus in (co)inductive type theory. *Theoretical Computer Science 239(2)*, 2001.
4. 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.
5. M. Miculan. Encoding logical theories of programs. PhD thesis, Dipartimento di Informatica, Universit`a di Pisa, Italy, 1997.
6. 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.
7. 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.
8. F. Pfenning and C. Schu¨rmann. System description: Twelf, a meta-logical framework for deductive systems. In *Proc. of CADE*, *Lecture Notes in Artiﬁcial Intelligence 1632*, 1999.
9. I. Scagnetto. Reasoning about Names in Higher-Order Abstract Syntax. Cs 2002/4, Dipartimento di Matematica e Informatica, Universit`a di Udine, 2002.