Reasoning on an Imperative Object-based Calculus in Higher Order Abstract Syntax

Alberto Ciaffaglione‡
ciaffagl@dimi.uniud.it Luigi Liquori† Luigi.Liquori@inria.fr Marino Miculan‡ miculan@dimi.uniud.it

ABSTRACT
We illustrate the benefits of using Natural Deduction in combination with weak Higher-Order Abstract Syntax for formalizing an object-based calculus with objects, cloning, method-update, types with subtyping, and side-effects, in inductive type theories such as the Calculus of Inductive Constructions. This setting suggests a clean and compact formalization of the syntax and semantics of the calculus, with an efficient management of method closures. Using our formalization and the Theory of Contexts, we can prove formally the Subject Reduction Theorem in the proof assistant Coq, with a relatively small overhead.

Categories and Subject Descriptors
D.3.1 [Programming Languages]: Formal definitions and Theory—syntax, semantics; F.3.1 [Specifying and Verifying and Reasoning about Programs]: [logics of programs]

General Terms
Languages, Theory, Verification.

Keywords
Interactive theorem proving, Logical foundations of programming, Program and system verification, Object-based calculi with side-effects, Logical Frameworks.

1. INTRODUCTION
Object-based languages, like Self and Obliq, are a kind of object-oriented languages where there is no notion of “class”, differently from class-based languages, like Java, C++ and C#.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MERLIN’03, August 26, 2003, Uppsala, Sweden
Copyright 2003 ACM 1-58113-800-8/03/0008 ...$5.00.

‡Research supported by Italian MIUR project COFIN 2001013518 COMETA.
†INRIA-LORIA, Nancy, France.
‡Department of Mathematics and Computer Science, University of Udine, Italy.
ject calculi; therefore, we will not discuss coinductive types and related proof systems. We refer the reader to [6].

Natural Deduction leads us to consider a different formulation of static and dynamic semantics of imp,$^\$C$, namely in the style of the Natural Deduction Semantics (NDS) [4, 22]. The key point in using NDS is that all stack-like structures (e.g., environments, typing bases) are distributed in the hypotheses of the proof derivations. Therefore, these structures do not appear explicitly anymore in the formal judgments and proofs we have to deal with, which become fairly simpler. Of course, the semantics in NDS we focus on is equivalent to the original Natural Semantics of [2].

On the other hand, the imp,$^\$C$ calculus has binders, which we have to deal efficiently with both in the syntax and in the semantics (due to the presence of closures). To this end, the weak Higher-Order Abstract Syntax (weak HOAS) approach has emerged as one of the most suited in inductive type theories like CIC [10, 17, 18]. Following such an approach, a separate, open type for variables is introduced, and binders are represented as second-order term constructors, taking functions over variables as arguments. In this way, it is possible to delegate to the metalanguage all the burden of $\alpha$-conversion, still retaining the benefits of inductive definitions. The disadvantage of weak HOAS is that it does not allow to inherit substitution of terms for variables from the metalanguage, as it is possible, instead, using full HOAS in non inductive type theories [15, 26]. Luckily, this is not a problem in the case of imp,$^\$C$, since its semantics is defined using closures instead of substitutions. Therefore, the solution of using weak HOAS fits perfectly the needs for encoding imp,$^\$C$ as a byproduct, the NDS and HOAS style yield a different, more efficient handling of closures.

However, the weak HOAS reveals its main drawback when we come to reasoning over our formalization of imp,$^\$C$, e.g., for proving key metaproperties such as the Subject Reduction. Here we have to prove several properties concerning variable renaming, often by induction over second-order terms, and by case analysis over the name of the variables themselves. All this is problematic because CIC, and similar type theories, are not expressive enough [18].

In recent years, various approaches have been proposed to overcome these kinds of problems. One approach tends to propose new, more expressive metalanguages, especially designed for reasoning on names and variables, and based on different techniques such as modal types, functor categories, permutation models of set theory, etc. [11, 12, 14, 16].

Another line of research tends to cope with these problems within the current systems and logical frameworks [18, 25, 27]. A comparison of the different approaches is out of the scope of this paper.

Since we carry out a weak HOAS encoding in CIC, we have decided to adopt the Theory of Contexts (ToC) [17], a quite general “plug-and-play” methodology for reasoning on object systems in weak HOAS. The gist of this approach is to extend directly the existing framework (CIC, in this case) with a small set of axioms capturing some basic and natural properties of names and term contexts. The main advantage of such a technique is that it requires a very low mathematical and logical overhead: it can be easily used in existing proof environments without the need of any redesign of the system, and it allows for carrying out proofs with arguments similar to those “on paper”. It turns out that also in the present case, the ToC reveals to be well-suited: we actually succeed in proving the Subject Reduction for imp,$^\$C$ with a small, sustainable overhead.

Our effort is useful also from the point of view of Logical Frameworks. Their theoretical development and implementation will benefit from complex case studies like the present one, where we test the applicability of advanced encoding and proof methodologies. An interesting remark is that we have to slightly modify the “unsaturation”, one of the axioms of the ToC, in order to take into account the different kinds of informations kept in the proof derivation environment. This is similar to previous applications of the ToC to typed languages [23], and it is needed in order to respect a form of “regularity” of well-formed contexts. Thus, we argue that such a “regularity” of proof contexts should allow to generalize further the ToC, subsuming the several variants used in the case studies so far.

Synopsis. The paper is structured as follows. Section 2 gives a brief account of the imp,$^\$C$-calculus, whose formalization in weak HOAS is presented in Section 3. Development of metatheoretic results, using the ToC, is discussed in Section 4. Final discussions and directions for future work are in Section 5. The Coq code is available at [7].

2. THE imp,$^\$C$ CALCULUS

Syntax. Abadi and Cardelli’s imp,$^\$C$-calculus [2] is an imperative calculus of objects forming the kernel of the programming language Obliq [5]. The syntax is the following:

$$\text{Term} : a, b ::= x \quad \text{variable}$$

$$[l_i = \varsigma(x_i)b_i]_{i\in I} \quad \text{object}$$

$$\text{clone}(a) \quad \text{cloning}$$

$$a.l \quad \text{method invocation}$$

$$a.l \leftarrow \varsigma(x)b \quad \text{method update}$$

$$\text{let } x = a \text{ in } b \quad \text{local declaration}$$

Notice that $\varsigma$ and let bind $x$ in $b$, so usual conventions about $\alpha$-conversion apply. The reader is referred to [2, Ch.10,11] for an explanation of the intuitive meaning of these constructions. We present the semantics of imp,$^\$C$ using a Natural Deduction (NDS) style, because this leads to a simpler formalization in LFs, as will be explained and carried out in the next section.

Dynamic Semantics. First we need to introduce some implementation-level entities, such as store-locations, results, reduction contexts, closures and stores:

$$\text{Loc} : \ i \in \text{Nat} \quad \text{store loc.}$$

$$\text{Res} : \ v ::= [l_i = \varsigma(x_i)b_i]_{i\in I} \quad \text{result}$$

$$\text{RedCtx} : \Gamma ::= \emptyset \mid \Gamma, x \mapsto v \ (x \notin \text{Dom}(\Gamma)) \quad \text{red. ctx.}$$

$$\text{Store} : \ s ::= \varsigma(x_i)b_i, I_{i\in I} \quad \text{store}$$

A method-closure $(\varsigma(x)b, \Gamma)$ is a method together with a set of declarations for its free variables, the (reduct) context. In reduction contexts, variables are uniquely associated to results of object declarations. Each result is a (possibly empty) list of pairs: method-names together with store-locations, where the corresponding method-closure is stored. Hence a store is a function mapping locations to closures: in the following, the notation $\varsigma(x_i)b_i, I_{i\in I}$ denotes the store that
maps the locations \(i\) to the closures \(c\), for \(i \in I\); \(s, t \mapsto c\) extends \(s\) with \(c\) at location \(i\) (fresh), and \(s, j \mapsto c\) denotes the result of replacing the content of the location \(i\) of \(s\) with \(c\). Unless explicitly remarked, all the \(i, j, t\) are distinct.

The big-step operational semantics is expressed by a reduction judgment relating a store \(s\), a term \(a\), a result \(v\) and another store \(s'\) (possibly different from \(s\)), in a given reduction context, i.e.:

\[ \Gamma \vdash s \cdot a \leadsto v \cdot s' \]

The intended meaning is that, evaluating the term \(a\) from the store in the reduction context \(\Gamma\), yields the result \(v\), in an updated store \(s'\). In this semantics, variables never need to be replaced (i.e. substituted) by terms; instead, their result is book-kept in the reduction context. The rules for the reduction judgment are in Figure 1. Slightly different from the original presentation [2], ours is in Natural Deduction style (despite that rules are written in “horizontal”, sequent-like format for saving space): in fact, in all the rules, the reduction context in each premise is a superset of the reduction context in the conclusion. This alternative presentation is much easier to implement and reason about in proof assistants based on type theory, as we will see in the next sections. Clearly, our presentation is equivalent to the original one (see [8,9]).

**Static Semantics.** The \(\text{imp}_\mathcal{C}\)-calculus has a first-order type system with subtyping. The only type constructor is that for object-types, i.e. \(\text{OType} : \ A, B ::= \ [i : A_i]^{\epsilon \in \mathcal{E}}\), so the only ground type is \([\ ]\).

In order to define the typing judgment we need to introduce the typing contexts (or bases), which are unique assignments of types to variables:

\[ \text{TypCtx} : \ \Delta ::= \emptyset \mid \Delta, x:A \quad (x \notin \text{Dom}(\Delta)) \]

The type system is given by three judgments: well-formedness of object-types \(\Delta \vdash A\), subtyping \(\Delta \vdash A < B\), and term typing \(\Delta \vdash a : A\). The rules for these judgments are collected in Figure 2; notice that also this system is in Natural Deduction style. Subtyping induces subsumption: an object of a given type also belongs to any super-type of that type and can subsume objects in the super-type, because they have a more limited protocol. Correspondingly, the rule (\(\text{Sub—Obj}\)) allows a longer object-type to be a subtype of a shorter one, when the shared components have exactly the same types. Thus, object-types are invariant (i.e. neither covariant nor contravariant) in their component types.

The typing of results is delicate, because results contain references to objects and methods. A clean and compact encoding is a prerequisite for simplifying the development of complex metatheoretical results, such as the Subject Reduction. In general, a naïve encoding would lead to a clumsy and unmanageable set of definitions, which would add further difficulties to the formal proofs, which are already difficult on paper. Therefore, we need encoding methodologies that take most advantage of the features provided by modern type theories, so that most (if not all) details implicitly taken for granted working with paper and pencil can be automatically inherited in the formal developments.

Among other issues, the treatment of binders, closures and stores is of particular interest. We focus on these aspects at the syntactic and semantic level, in turn.

**3. FORMALIZATION OF \text{imp}_\mathcal{C}\ IN \ CIC**

In this section, we present and discuss the formalization of \(\text{imp}_\mathcal{C}\) in our \(\text{LF}\) based on type theory. For definiteness, we work in the Calculus of Inductive Constructions (CIC), and specifically in its \(\text{Coq}\) implementation, although the methodology we follow can be applied in any similar \(\text{LF}\).

In encoding \(\text{imp}_\mathcal{C}\), we have to address efficiently several aspects of its syntax and semantics. A clean and compact encoding is a prerequisite for simplifying the development of complex metatheoretical results, such as the Subject Reduction.

**3.1 Syntax**

Following the weak higher-order abstract syntax paradigm, we reduce all binders to the sole \(\lambda\)-abstraction. Therefore, the encoding of the syntax of terms and types of \(\text{imp}_\mathcal{C}\) is the following:

Parameter \(\text{Var} : \text{Set}\).

Inductive \(\text{TType} : \text{Set} \to \text{Set} \to \text{Term}\) with:

1. \(\text{obj} : \text{Object} \to \text{Term}\)
2. \(\text{call} : \text{Term} \to \text{Lab} \to \text{Term}\)
3. \(\text{clone} : \text{Term} \to \text{Term}\)
4. \(\text{over} : \text{Term} \to \text{Lab} \to (\text{Var} \to \text{Term}) \to \text{Term}\)
5. \(\text{let} : \text{Term} \to (\text{Var} \to \text{Term}) \to \text{Term}\)

with \(\text{Object} : \text{Set} := \text{obj Nil} : \text{Object}\)

and for all \(\text{Var} \to \text{Term} \to \text{Object} \to \text{Object}\).

Coercion \(\text{var} : \text{Var} \to \text{Term}\).

Inductive \(\text{TTType} : \text{Set} \to \text{TTType} = \text{mkttype} : ([\text{list} (\text{Lab} \times \text{TTType})) \to \text{TTType})\).
3.2 Dynamic Semantics

As pointed out in the previous sections, a proof system in natural deduction like that in Figure 1 is easily encoded in a LF based on type theory. This can be carried out by taking advantage of hypothetical-general judgments à la Martin-Löf; the content of Γ, i.e., the bindings between variables and results, is actually represented through assumptions in the proof context, namely by means of hypothetical premises local to sub-reductions, which are discharged in the conclusion, so adhering to the natural deduction style. Thus, the evaluation judgment Γ ⊢ s · a ∼ v · s′ can be represented by an inductive predicate:

\[
\begin{align*}
\text{Inductive } \text{eval} & : \text{Store} \to \text{Term} \to \text{Store} \to \text{Res} \to \text{Prop} \Rightarrow \\
\text{but, before giving its specification, we need to add some data} &
\end{align*}
\]

Notice that we use a separate type Var for variables: the only terms which can inhabit Var are the variables of the metalanguage. Thus α-equivalence on terms is immediately inherited from the metalanguage, still keeping induction and recursion principles. If Var were inductive, we could define “exotic terms” using the Case construct of Coq [10]. Exotic terms are Coq terms not corresponding to any expression of the metalanguage, and thus may omitted in the following. Weak HOAS has well-known encoding methodologies [15, 28]; therefore, the adequacy of the encoding w.r.t. the NDS presentation of Section 2 follows from standard arguments.

An alternative definition of objects could use directly the polymorphic lists of the Coq library, instead of introducing a separate type Object, as follows:

In our experience, this choice would not allow for defining, by recursion on the structure of terms, some fundamental functions which are essential for the rest of the formalization (such as, for example, the non-occurrence of variables \( \epsilon \)). Using polymorphic lists, the specification of these functions would be actually “unguarded”, while they are feasible adopting the above definition.

3.2 Dynamic Semantics

As pointed out in the previous sections, a proof system in natural deduction like that in Figure 1 is easily encoded in a LF based on type theory. This can be carried out by taking advantage of hypothetical-general judgments à la Martin-Löf; the content of Γ, i.e., the bindings between variables and results, is actually represented through assumptions in the proof context, namely by means of hypothetical premises local to sub-reductions, which are discharged in the conclusion, so adhering to the natural deduction style. Thus, the evaluation judgment Γ ⊢ s · a ∼ v · s′ can be represented by an inductive predicate:

\[
\begin{align*}
\text{Inductive eval} & : \text{Store} \to \text{Term} \to \text{Store} \to \text{Res} \to \text{Prop} \Rightarrow \\
\text{but, before giving its specification, we need to add some data} &
\end{align*}
\]

Notice that we use a separate type Var for variables: the only terms which can inhabit Var are the variables of the metalanguage. Thus α-equivalence on terms is immediately inherited from the metalanguage, still keeping induction and recursion principles. If Var were inductive, we could define “exotic terms” using the Case construct of Coq [10]. Exotic terms are Coq terms not corresponding to any expression of the metalanguage, and thus may omitted in the following. Weak HOAS has well-known encoding methodologies [15, 28]; therefore, the adequacy of the encoding w.r.t. the NDS presentation of Section 2 follows from standard arguments.

An alternative definition of objects could use directly the polymorphic lists of the Coq library, instead of introducing a separate type Object, as follows:

In our experience, this choice would not allow for defining, by recursion on the structure of terms, some fundamental functions which are essential for the rest of the formalization (such as, for example, the non-occurrence of variables \( \epsilon \)). Using polymorphic lists, the specification of these functions would be actually “unguarded”, while they are feasible adopting the above definition.

3.2 Dynamic Semantics

As pointed out in the previous sections, a proof system in natural deduction like that in Figure 1 is easily encoded in a LF based on type theory. This can be carried out by taking advantage of hypothetical-general judgments à la Martin-Löf; the content of Γ, i.e., the bindings between variables and results, is actually represented through assumptions in the proof context, namely by means of hypothetical premises local to sub-reductions, which are discharged in the conclusion, so adhering to the natural deduction style. Thus, the evaluation judgment Γ ⊢ s · a ∼ v · s′ can be represented by an inductive predicate:

\[
\begin{align*}
\text{Inductive eval} & : \text{Store} \to \text{Term} \to \text{Store} \to \text{Res} \to \text{Prop} \Rightarrow \\
\text{but, before giving its specification, we need to add some data} &
\end{align*}
\]

Notice that we use a separate type Var for variables: the only terms which can inhabit Var are the variables of the metalanguage. Thus α-equivalence on terms is immediately inherited from the metalanguage, still keeping induction and recursion principles. If Var were inductive, we could define “exotic terms” using the Case construct of Coq [10]. Exotic terms are Coq terms not corresponding to any expression of the metalanguage, and thus may omitted in the following. Weak HOAS has well-known encoding methodologies [15, 28]; therefore, the adequacy of the encoding w.r.t. the NDS presentation of Section 2 follows from standard arguments.

An alternative definition of objects could use directly the polymorphic lists of the Coq library, instead of introducing a separate type Object, as follows:

In our experience, this choice would not allow for defining, by recursion on the structure of terms, some fundamental functions which are essential for the rest of the formalization (such as, for example, the non-occurrence of variables \( \epsilon \)). Using polymorphic lists, the specification of these functions would be actually “unguarded”, while they are feasible adopting the above definition.

3.2 Dynamic Semantics

As pointed out in the previous sections, a proof system in natural deduction like that in Figure 1 is easily encoded in a LF based on type theory. This can be carried out by taking advantage of hypothetical-general judgments à la Martin-Löf; the content of Γ, i.e., the bindings between variables and results, is actually represented through assumptions in the proof context, namely by means of hypothetical premises local to sub-reductions, which are discharged in the conclusion, so adhering to the natural deduction style. Thus, the evaluation judgment Γ ⊢ s · a ∼ v · s′ can be represented by an inductive predicate:

\[
\begin{align*}
\text{Inductive eval} & : \text{Store} \to \text{Term} \to \text{Store} \to \text{Res} \to \text{Prop} \Rightarrow \\
\text{but, before giving its specification, we need to add some data} &
\end{align*}
\]
\[
\frac{
\Delta \vdash A \quad \forall i \in I \\
\Delta \vdash \text{let } x = a \text{ in } b : B \\
\Delta \vdash x : A}
}{
\Delta \vdash \text{let } x = a \text{ in } b : B}
\]

**Figure 2:** Type assignment system for terms of impc.

\[
\frac{
\Delta \vdash A < : B \\
\Delta \vdash B < : C \\
\Delta \vdash A < : C}
}{
\Delta \vdash A < : C}
\]

\[
\frac{
\Delta \vdash a : A \\
\Delta \vdash A < : B}
}{
\Delta \vdash a : B}
\]

\[
\frac{
\Delta, x : \text{let } \gamma(x)b : \text{let } \delta \in I \\
\Delta \vdash \text{let } \gamma(x)b : \text{let } \delta \in I \\
\Delta \vdash \text{let } \gamma(x)b : \text{let } \delta \in I}
}{
\Delta \vdash \text{let } \gamma(x)b : \text{let } \delta \in I}
\]

\[
\frac{
\Delta \vdash a : \text{let } \gamma(x)b : \text{let } \delta \in I \\
\Delta \vdash \text{let } \gamma(x)b : \text{let } \delta \in I \\
\Delta \vdash \text{let } \gamma(x)b : \text{let } \delta \in I}
}{
\Delta \vdash \text{let } \gamma(x)b : \text{let } \delta \in I}
\]

**Figure 3:** Type assignment system for results of impc.

Inductive `Body` : `Set` := `ground` : `Term` -> `Body`  
| `bind` : `Res` -> `(Var->Body)` -> `Body`.

**Definition Closure**  : `Set` := `Var` -> `Body`.

For instance, the encoding of the closure
\[
\langle \gamma(x) b, \{ x_1 \mapsto v_1, \ldots, x_n \mapsto v_n \} \rangle
\]
is the following term of type `Closure`:

\[
[\text{x:Var}] (\text{bind } v_1 \\
[\text{x:Var}] \ldots (\text{bind } v_n \\
[\text{xn:Var}] (\text{ground b})) \ldots)
\]

where the outermost abstraction represents the "self" local variable `x`. Since stores are finite maps from locations to closures, we can define stores straightforwardly as:

**Definition Store** : `Set` := `(list Closure)`.

Some functions are needed for manipulating the above structures (e.g., for merging lists of pairs into single lists, for generating new results from objects and results, etc.), but their definition is not problematic; see [7] for the code.

**Evaluation of closures.** Let us see how closures are "opened" in the implementation of the rule for method invocation (`Red-Set`). Essentially, we have to add to the current proof environment all the bindings recorded in the closure; then, the evaluation of the closure-body may take place in the extended environment. The evaluation of closure-bodies can be pre-formalized by an auxiliary judgment \( \Gamma \vdash \Delta \vdash A \vdash A < : A \), whose rules are the following:

\[
\begin{align*}
\Gamma \vdash s \cdot a \sim v \cdot s' & \quad (c_{\text{ground}}) \\
\Gamma \vdash s \cdot \text{ground}(a) \sim_{\text{h}} s' \cdot v \\
\Gamma \vdash x \mapsto v \cdot s \cdot \tilde{b} \sim_{\text{h}} v' \cdot s' & \quad (c_{\text{bind}}) \\
\Gamma \vdash s \cdot \text{bind}(v, \lambda y. \tilde{b}) \sim_{\text{h}} v' \cdot s'
\end{align*}
\]

Correspondingly, rule `(Red-Set)` can be stated better as:

\[
\begin{align*}
\Gamma, \Gamma' \vdash x_j \mapsto [\text{let } \gamma(x)b : \text{let } \delta \in I \\
\Gamma, \Gamma' \vdash x_j \mapsto [\text{let } \gamma(x)b : \text{let } \delta \in I \\
\Gamma, \Gamma' \vdash x_j \mapsto [\text{let } \gamma(x)b : \text{let } \delta \in I \\
\Gamma \vdash s \cdot \text{bind}(v, \lambda y. \tilde{b}) \sim_{\text{h}} v' \cdot s'' & \quad (Red-Set')
\end{align*}
\]
Thus, \( \sim \) and \( \sim_b \) are two mutually recursive predicates:

**Mutual Inductive**

\[
\text{eval} : \text{Term} \to \text{Term} \to \text{Store} \to \text{Res} \to \text{Prop} \\
\text{eval} : \text{Body} \to \text{Store} \to \text{Term} \to \text{Prop} \\
\]

The rules for `eval_body` are immediate; let us see the encoding of the above \( \text{Red-Sel} \) of the latter can be managed more efficiently as a function formalized by two inductive predicates, as usual. However, in a method where the variable more free variables to bind) and thus apply rule (of the above \( \text{Red-Sel} \)) goal is easily dealt with using the tactics provided by Coq. In a similar way, we define also the functions \text{notin}: \text{Var} \to \text{Term} \to \text{Prop} and \text{fresh}: \text{Var} \to \text{list Var} \to \text{Prop}, which capture the “freshness” of a variable in a term and \( w.r.t \) a list of variables, respectively (see [7]). Finally, the judgment \text{wrap} is formalized via an inductive predicate:

\[
\text{Inductive wrap} : \text{Term} \to \text{Body} \to \text{Prop} \\
\text{wrap} : \text{Term} \to \text{Body} \\
\]

**Construction of closures.** The construction of closures is more delicate. We cannot form pairs out of the proof context “\( \Gamma \)” and methods: closures have to be “calculated” by wrapping method-bodies with all the results associated to their free variables, which are in the proof context. This closure construction is carried out by the auxiliary judgment \text{wrap} \( \subseteq \text{Term} \times \text{Body} \). The intended meaning of \( \Gamma \vdash \text{wrap}(b, 5) \) is that “\( \bar{y} \)” is a closure-body obtained by binding all free variables in the term \( b \) to their respective results, which are in \( \Gamma \). In order to keep track of free variables in terms, we introduce a judgment \text{closed} \( \subseteq \text{Term} \), whose formal meaning is \( \Gamma \vdash \text{closed}(a) \iff \forall y \in \text{FV}(a): \text{closed}(x) \in \Gamma \). The rules for the two additional judgments are in Figure 4. Intuitively, the rules for \text{wrap} allow for successively binding the free variables appearing in the method-body \( \text{w_bind} \) until it is “closed” \( \text{w_ground} \). When we apply the rule \( \text{w_bind} \), we choose any \( \text{free variable} y \in b \), and bind it to the corresponding result \( y \), as stated in \( \Gamma \). The remaining part \( b \) of the closure can be seen as the closure body of a method where the variable \( z \) is supposed to be “closed”, and therefore it is obtained in a proof environment containing this information. This is captured by the sub-derivation \( \Gamma, \text{closed}(z) \vdash \text{wrap}(b[z/y], b[z/y]) \), which applies the rule \( \text{w_bind} \) until enough variables have been bound and, correspondingly, enough assumptions of the form \( \text{closed}(z) \) have been taken to be able to prove \( \text{closed}(b) \) (i.e., there are no more free variables to bind) and thus apply rule \( \text{w_ground} \). Notice that the closures we get in this manner are “optimized”, because only variables which are really free in the body need to be closed in the closure (although in a non-deterministic order).

The two auxiliary judgments \text{wrap} and \text{closed} can be formalized by two inductive predicates, as usual. However, the latter can be managed more efficiently as a function \text{closed}: \text{Term} \to \text{Prop}, in the style of [24]:

\[
\text{Parameter} \quad \text{dummy} : \text{Var} \to \text{Prop} \\
\text{Fixpoint} \quad \text{closed} \quad \text{t:Term} : \text{Prop} \Rightarrow \text{Cases} t \text{ of} \\
(\text{var} x) \Rightarrow (\text{dummy} x) | \\
(\text{obj} m) \Rightarrow (\text{closed_obj} m l) \\
| (\text{call} a 1) \Rightarrow (\text{closed} a) \\
| (\text{let} a b) \Rightarrow (\text{closed} a) \\
| (x:Var) \Rightarrow (\text{closed_obj} (\text{dummy} x)) \\
\]

with \text{closed_obj} \ (\text{ml:Object}) : \text{Prop} := \text{Cases} ml \text{ of} \\
(\text{obj_nil}) \Rightarrow \text{True} | (\text{obj_cons} l \text{ ml}) \Rightarrow \\
(\text{closed_obj} ml) \land \\
((x:Var) (\text{dummy} x)) \Rightarrow (\text{closed} (m x)) \\
\]

end.

The intended behavior of this function, defined by mutual recursion on the structure of terms and objects, is to reduce an assertion \( \text{closed} \) into a conjunction of simpler assertions about simpler terms. The \text{dummy} is the usual workaround for the negative occurrences of \text{closed} in the definition: \text{dummy} variables are just fill-ins for holes, and must be considered as “closed”. The proposition resulting from the Simplification of \( \text{closed} \) goal is easily dealt with using the tactics provided by Coq. In a similar way, we define also the functions \text{notin}: \text{Var} \to \text{Term} \to \text{Prop} and \text{fresh}: \text{Var} \to \text{list Var} \to \text{Prop}, which capture the “freshness” of a variable in a term and \( w.r.t \) a list of variables, respectively (see [7]). Finally, the judgment \text{wrap} is formalized via an inductive predicate:

\[
\text{Inductive wrap} : \text{Term} \to \text{Body} \to \text{Prop} \\
\text{wrap} : \text{Term} \to \text{Body} \\
\]

The typing of terms is defined by mutual induction with the typing of objects; notice only that we need to carry along the whole \( \text{object} \) type while we scan and type the list of methods forming the objects:

\[
\text{Mutual Inductive} \quad \text{type} : \text{Term} \to \text{FF} \to \text{Prop} \\
| t_{\text{sub}} : (\text{a:Term}; A; B; T\text{Type}) \Rightarrow (\text{type} A) \\
| (\text{sub} A B) \Rightarrow (\text{type} B) \\
\]

**Typing of terms.** The encoding of the typing system for terms is straightforward. Like done for the reduction context, we model the typing environment by means of a functional symbol, associating object-types to variables:

\[
\text{Parameter typeenv} : \text{Var} \to \text{FF} \\
\]

The typing of terms is defined by mutual induction with the typing of objects; notice only that we need to carry along the whole \( \text{object} \) type while we scan and type the list of methods forming the objects:

\[
\text{Mutual Inductive} \quad \text{type} : \text{Term} \to \text{FF} \to \text{Prop} \\
| t_{\text{sub}} : (\text{a:Term}; A; B; T\text{Type}) \Rightarrow (\text{type} A) \\
| (\text{sub} A B) \Rightarrow (\text{type} B) \\
\]
\[
\begin{array}{ll}
\Gamma \vdash \text{closed}(b) & (w_{\text{ground}}) \\
\Gamma \vdash \text{wrap}(b, \text{ground}(b)) & (c_{\text{clone}}) \\
\Gamma \vdash \text{closed}(a) & (c_{\text{over}}) \\
\Gamma \vdash \text{closed}(\text{clone}(a)) & (\neg e) \\
\end{array}
\]

\[
\begin{array}{ll}
\Gamma \vdash \text{closed}(a)\ \Gamma,\text{closed}(x) \vdash \text{closed}(b) & (c_{\text{let}}) \\
\Gamma \vdash \text{closed}(\text{let } x = a \ in \ b) & (w_{\text{bind}}) \\
\end{array}
\]

**Figure 4:** Rules for \text{wrap} and \text{closed} judgments.

\begin{itemize}
\item \text{t_obj} : (ml: Object) (A:TType) (type_obj A (obj ml) A) \rightarrow (type (obj ml) A) \rightarrow (\text{mctype (nil (Lab+TType))})
\item \text{t_nil} : (A:TType) (type_obj A (obj nil) (\text{mctype nil (Lab+TType)})) \rightarrow (\text{mktype (nil (Lab*TType))})
\item \text{t_step} : (S:SType) (A,B,C:TType) (v:Res) (i:Loc) (1:Lab) (pl:(list (Lab+TType))) (type_obj C (obj ml) A) \rightarrow ((x:Var) \ (\text{typenv x}) = (C) \rightarrow (type (n x) B) \rightarrow (\text{mktype (cons (l,B) pl)}))
\item \text{t_cons} : (A,B,C:TType; \ldots : n:Var\rightarrow Term; pl:(List (Lab+TType))) (type_obj C (obj ml) (\text{mctype (cons (l,B) pl)}))
\end{itemize}

where \text{sub} represents the subtype predicate ":<:". Due to lack of space, we omit here the encoding, which makes also use of an auxiliary predicate for permutation of lists representing object types (see [7]).

**Typing of results.** In order to type results, it is necessary to type store-locations, whose content, in turn, may contain other pointers to the store: thus potential loops may arise in this process. The solution adopted in [2] is to introduce yet another typing structure, i.e. store-types, used for assigning to each store location a type consistent with its content:

**Definition SType :** Set := (list (TType * TType)).

Store-types cannot be inferred from the stores, but have to be provided beforehand; they are used for assigning types to results by means of the auxiliary proof system of Figure 3. Such a system is easily rendered in CIC; we only point out that, similarly to the typing of objects, we have to carry along the whole (result) type while we scan and type the (components of) results:

\begin{itemize}
\item \text{Inductive res : SType \rightarrow TType \rightarrow Res \rightarrow TType \rightarrow Prop :=}
\item \text{t_void : (S:SType) (A:TType) (res S A (nil (Lab+Loc))) \rightarrow (\text{mctype nil (Lab+TType)}))}
\item \text{t_step : (S:SType) (A,B,C:TType) (v:Res) (i:Loc) (1:Lab) (pl:(list (Lab+TType))) (res S A v B) \rightarrow (\text{type_from_lab_1 B)} C \rightarrow (\text{type_nth_1 C) S) \rightarrow (\text{type nth_2 C) S) \rightarrow (\text{mctype (cons (l,C) pl)}))}
\end{itemize}

\begin{itemize}
\item \text{Inductive type body : SType \rightarrow Body \rightarrow TType \rightarrow Prop :=}
\item \text{t_ground : (S:SType) (b: Term) (A:TType) (type b A) \rightarrow (\text{type body S (ground b) A})}
\item \text{t_bind : (S:SType) (b: Var\rightarrow Body) (A,B:TType) (v:Res) (\text{type res S A v A}) \rightarrow ((x:Var) \ (\text{typenv x}) = (A) \rightarrow (\text{type body S (b x) B) \rightarrow (\text{type body S (bind v b) B)})}
\end{itemize}

**Definition ext : SType \rightarrow SType \rightarrow Prop := [S',S:SType] (le (dim S) (dim S')) \land (i:nat) (1 i (dim S)) \rightarrow (\text{type_nth i S') (\text{type_nth i S})).

**Definition comp :** SType \rightarrow SType \rightarrow Prop := [S:SType; s:Store] ((\text{type body S (store_nth i s x) A})) \rightarrow ((\text{type body S (store_nth i s x) A)})

The functions for store-type manipulation are in [7].

### 3.4 Adequacy

Our formalization of the semantics of \text{imp} in Coq corresponds faithfully to the NDS one of Section 2, as we see now (for the full proof, we refer the interested reader to [8,9]). First, we establish the relationship between our heterogeneous (Coq) context \(\Gamma\) and the reduction and typing environments \(\Gamma,\Delta\) of the setting of Section 2, and between the two kinds of stores \(\tau\) and \(s\). In the following, we write \(\Gamma + \tau\) for \(\exists !. \ \Gamma + \tau\) i.e.:

**Definition 2.** (Well-formed context). A context \(\Gamma\) is well-formed if and only if it can be partitioned as \(\Gamma = \Gamma_s, \Gamma_t, \Gamma_d\), where \(\Gamma_s = \ldots \text{stack} x_i = v_i, \ldots\) and \(\Gamma_t = \ldots t_i : (\text{typenv x}_i) = \text{b}_i, \ldots\) and \(\Gamma_d = \text{d}_i : (\text{dummy x}_i), \ldots\) moreover, \(\Gamma_s\) and \(\Gamma_t\) are functional (e.g., if \(\text{stack}(x) = v\), \(\text{stack}(x') = v'\) in \(\Gamma_s\), then \(v \equiv v'\)).

**Definition 3.** Let \(\Gamma\) be a context, \(\Gamma, \Delta\) reduction and typing contexts, \(\tau\) and \(s\) stores. We define:

\(\begin{align*}
\Gamma & \subseteq \Delta \quad \forall (\text{stack} x) = v \in \Gamma, \ x \mapsto v \in \Gamma \\
\Gamma & \subseteq \Delta \quad \forall (\text{typenv x}) = A \in \Delta, \ x:A \in \Delta \\
\Gamma & \subseteq \Delta \quad \forall x:v \in \Gamma, \ (\text{stack} x) = v \in \Gamma \\
\Gamma & \subseteq \Delta \quad \forall x:A \in \Delta, \ (\text{typenv x}) = A \in \Delta \\
\gamma(\Gamma) & \subseteq \Gamma \quad \forall (\text{stack} x) = x \in \text{Dom}(\Gamma) \\
\tau & \subseteq s \quad \forall s \in \text{Dom}(\tau), \ \gamma(\Gamma), \\
\text{dummy x}_i & \mapsto (\text{wrap b}_i, (\text{tr x}_i), \text{x}_i), \text{where} \ s(x_i) = (\langle \text{x}_i, b_i, \gamma(\Gamma) \rangle) \\
\end{align*}\)

In the following, for \(b\) a closure-body, let us denote by \(\text{stack}(b)\) the reduction context containing the bindings in \(b\), and by \(\text{body}(b)\) the innermost body. These functions can be defined
by recursion:

\[
\begin{align*}
\text{stack}(\text{ground}(b)) & \triangleq \emptyset \\
\text{stack}(\text{bind}(v, \lambda x.b)) & \triangleq \text{stack}(b) \cup \{x \mapsto v\} \\
\text{body}(\text{ground}(b)) & \triangleq b \\
\text{body}(\text{bind}(v, \lambda x.b)) & \triangleq \text{body}(b)
\end{align*}
\]

**Theorem 2** (Adequacy of Reduction). Let \( \Sigma \) be well-formed, \( \Gamma \) a reduction context.

1. Let \( \Delta \subseteq \Gamma \), and \( \tau \not\subseteq s \).
   
   \( \Sigma, \Gamma \vdash (\text{eval}\;\tau\;a\;\tau'\;v) \Rightarrow \exists s'. \)
   
   \( \Gamma \vdash s \cdot a \leadsto v \cdot s', \) and \( \tau' \not\subseteq s' \).

2. Let \( \Gamma \subseteq \Gamma \), and \( s \not\subseteq \tau \).
   
   \( \Gamma \vdash s \cdot a \leadsto v \cdot s \Rightarrow \exists \tau' \).
   
   \( \Sigma, \Gamma \vdash (\text{eval}\;\tau\;a\;\tau'\;v) \).

**Proof.**

1. The two points are proved by mutual structural induction on the derivations of
   
   \( \Sigma, \Gamma \vdash (\text{eval}\;\tau\;a\;\tau'\;v) \) and \( \Sigma, \Gamma \vdash (\text{eval}\;\text{body}\;\tau\;b\;\tau'\;v) \).

2. By structural induction on the derivation of \( \Gamma \vdash s \cdot a \leadsto v \cdot s' \).

**Theorem 3** (Adequacy of Term Typing). Let \( \Sigma \) be well-formed, \( \Delta \) a typing context.

1. If \( \Sigma \subseteq \Delta \), and \( \Gamma \vdash (\text{type}\;a\;A) \), then \( \Delta \vdash a : A \).

2. If \( \Delta \subseteq \Sigma \), and \( \Delta \vdash a : A \), then \( \Gamma \vdash (\text{type}\;a\;A) \).

**Proof.**

1. By structural induction on the derivation of \( \Gamma \vdash (\text{type}\;a\;A) \).

2. By structural induction on the derivation of \( \Delta \vdash a : A \).

**Theorem 4** (Adequacy of Result Typing). Let \( \Sigma \) be well-formed, and \( \Sigma \) such that \( \Sigma \vdash \alpha \).

1. For \( \tau \not\subseteq s \), if \( \Sigma \vdash (\text{res}\;\Sigma\;v\;a) \) and \( \Sigma \vdash (\text{comp}\;\Sigma\;\tau) \), then \( \Sigma \vdash v : A \) and \( \Sigma \vdash s \).

2. For \( s \not\subseteq \tau \), if \( \Sigma \vdash v : A \) and \( \Sigma \vdash s \), then \( \Sigma \vdash (\text{res}\;\Sigma\;v\;a) \) and \( \Sigma \vdash (\text{comp}\;\Sigma\;\tau) \).

**Proof.**

1. By induction on \( v \) and inspection on the derivation \( \Sigma \vdash (\text{comp}\;\Sigma\;\tau) \).

2. By inspection on the derivations of \( \Sigma \vdash v : A \) and \( \Sigma \vdash s \).

4. **METATHEORY OF IMP\(\_\) IN Coq**

One of the main aims of the formalization presented in the previous section is to allow for the formal development of important properties of imp. In this section we discuss briefly the uppermost important, yet delicate to prove, metaproperty of the Subject Reduction (Theorem 1). Its formalization in Coq is as follows:

**Theorem SR:**

\[
\begin{align*}
(s, t : \text{Store}) & \vdash (a : \text{Term}) (v : \text{Res}) \\
(\text{eval} \; s \cdot a \; v) & \Rightarrow (A : \text{Type}) (x : A) \Rightarrow \\
& (S : \text{Type}) (\text{comp} \; S \; s) \Rightarrow \\
& ((x : \text{Var} \cdot v \cdot \text{Res} : C : \text{Type}) (\text{stack} \; x) = (u) \land \\
& (\text{typenv} \; x) = C \Rightarrow (\text{res} \; S \; w \; C) \Rightarrow \\
& (\text{EX} \; B : \text{Type} \mid (\text{res} \; B \; v \; B) \land (\text{ext} \; T \; S) \land \\
& (\text{comp} \; T \; t) \land (\text{sub} \; B \; A))).
\end{align*}
\]

The proof is by induction on the derivation of \( (\text{eval} \; s \cdot a \; v) \). In order to prove the theorem, we have to address preliminarily all the specific aspects concerning the management of concrete structures, as objects, object-types, results, stores, store-types, and so on: thus, many technical lemmata about operational semantics, term and result typing have been specified and formally proved. It turns out that these lemmata are relatively compact to prove, due to the setting of our formalization, which is centered around the natural deduction style of proof and, correspondingly, the use of HOAS.

Natural Deduction allows to distribute stack-like structures (i.e., reduction and typing contexts) in the hypotheses of the proof derivations: therefore, judgments and proofs we have to deal with are appreciably simpler than traditional ones. Weak HOAS, as previously remarked, fits perfectly the needs for encoding imp: on one hand, it allows to get the \( \alpha \)-equivalence for free; on the other hand, the fact that the general form of substitution is not automatically provided by weak HOAS is not a problem, because it is not required by the semantics of imp.

However, it is well-known that HOAS presents some drawbacks: the main one is that most LFs do not provide an adequate support for higher-order encodings. For example, these systems neither provide reduction/induction principles over higher-order terms (i.e., terms with “holes”) nor allow to access the notions related to the mechanisms delegated to the metalanguage. An important family of properties which cannot be proved in plain CIC are the so-called renaming lemmata, that is, invariance of validity under variable renaming, such as the following, regarding types of terms and closure-bodies of imp:

**Lemma rename\_term:**

\[
\begin{align*}
& (m : \text{Var} \rightarrow \text{Term}) (A : \text{Type}) (x, y : \text{Var}) \\
& (\text{type}\;a\;A) \Rightarrow \\
& (\text{typenv}\;x) = (\text{typenv}\;y) \Rightarrow \\
& (\text{type}\;a\;A) \Rightarrow \\
& (\text{body}\;a) (\text{body}\;b) = (\text{body}\;b) (\text{body}\;a) \Rightarrow \\
& (\text{comp}\;T\;t) \Rightarrow \\
& (\text{sub} \; B \; A) \Rightarrow \\
& (\text{res} \; T \; B \; v \; B) \land (\text{ext} \; T \; S) \land \\
& (\text{comp} \; T \; t) \land (\text{sub} \; B \; A)).
\end{align*}
\]

In other words, the expressive power of LFs is limited, when it comes to reason on formalizations in (weak) HOAS.

A simple and direct way for recovering the missing expressive power is by assuming a suitable (but consistent) set of axioms. This is the approach adopted in [17, 28], where ToC, a simple axiomatization capturing some basic and natural properties of (variable) names and term contexts, is proposed. These axioms allow for a smooth handling of schemata in HOAS, with a very low mathematical and logical overhead, hence they can be plugged in existing proof environments without requiring any redesign of the system. Their usefulness has been demonstrated in several case studies regarding untyped and simply typed \( \lambda \)-calculus, \( \pi \)-calculus, and Ambients [18, 23, 29]. Therefore, the use of the ToC is a natural choice also in the present setting; it should be noticed that the present one is the first application of the ToC to an object-based calculus. The Theory of Contexts consists in four axioms (indeed, axiom schemas):

unsaturation: \( \forall M. \; \exists x. \; x \in \text{FV}(M) \).

This axiom captures the intuition that, since terms are finite entities, they cannot contain all the variables at once. The same axiom can be stated w.r.t. lists of variables, instead of terms, since it is always possible to obtain from a term the list of its free variables: \( \forall L. \; \exists x. \; x \notin L \)
decidability of equality over variables: \(\forall x, y. x = y \lor x \neq y\). In a classical framework, this axiom is just an instance of the Law of Excluded Middle. In the present case, it represents the minimal classical property we need in an (otherwise) intuitionistic setting;

\[\beta\text{-expansion: } \forall M, x. 3N(\cdot), x \notin FV(N(\cdot)) \land M = N(x).\]

Essentially, \(\beta\text{-expansion}\) allows to generate a new context \(N\) with one more hole from another context \(M\), by abstracting over a given variable \(x\);

\[\text{extensionality: } \forall M(\cdot), N(\cdot), x. x \notin FV(M(\cdot), N(\cdot)) \land (M(x) = N(x)) \implies M(\cdot) = N(\cdot).\]

Extensionality allows to conclude that two contexts are equal if they are equal when applied to a fresh variable \(x\). Together with \(\beta\text{-expansion}\), this allows to derive properties by reasoning over the structure of higher-order terms.

The above axioms are very natural and useful for dealing with higher-order terms as methods, closures and local declarations. An important remark is that, in order to be effectively used for reasoning on impc, the “unsaturation” axiom has to be slightly modified with respect to its original formulation in [17].

One first difference is due to the presence of types. Similarly to the case of other typed languages, we assume informally that there are infinite variables for every type. This is equivalent to say that each variable “generated” by the unsaturation axiom can be associated to a given type.

A second difference, peculiar to impc and not required by any previous application of the ToC, is due to the presence of implementation-level entities, namely closures. In the formalization of impc, presented in the previous section, variables are introduced in the Coq derivation context for two reasons: either during reductions and typing assignments (associated respectively to results and types), or in the construction of closures (used just as place-holders). In the first case the new variable is associated both to results and types by the \(\text{stack}\) \(\text{typenv}\) maps. In the second case, the new variable is marked as \(\text{dummy}\), because it does not carry any information about results, but is used just as a typed place-holder, needed for typing closure-bodies in the store.

Thus, we observe a kind of “regularity” of well-formed contexts: for each variable \(x\), there is always the assumption \((\text{typenv } x) = \alpha\) for some \(\alpha\), and, either \((\text{stack } x) = \nu\) for some \(\nu\), or \((\text{dummy } x)\). The unsaturation axiom has to respect such a regularity; this is reflected by assuming two variants of unsaturation, one for each case, as follows:

**Axiom unsat_res :** \((S:\text{SType}) \land (v:\text{Res}) \land (A:\text{TType}) \land (\text{res } S \land A)\rightarrow (x :\text{List } Var)\rightarrow (\text{EX } x | (\text{fresh } x \land x) \land (\text{stack } x) = \nu \land (\text{typenv } x) = A).\)

**Axiom unsat :** \((A:\text{TType}) \land (x:\text{List } Var)\rightarrow (\text{EX } x | (\text{fresh } x \land x) \land (\text{typenv } x) = A).\)

In \text{unsat_res}, the premise \((\text{res } S \land A)\) ensures the consistency between results and types associated to the same variable: this can be interpreted as the implementation of the original context compatibility judgment of Figure 3.

The two axioms can be validated in models similar to those of the original ToC [3]. Moreover, the notion of regularity is close to the “regular world assumption” introduced by Schürmann [30].

In order to convey better to the reader the above explanation about the ToC, we say that a typical example of the use of the axiom \text{unsat} is for proving that the type of a closure-body is preserved by the closure construction:

Lemma \text{wrap_pres_type : } (A,B:\text{Type}) \land (m:\text{Var}\rightarrow \text{Term}) \land (c:\text{Closure}) \land (x:1:\text{List } Var) \land (S:\text{SType}) \land ((x:Var) \land (\text{typenv } x) = A \rightarrow (\text{type } (m x) B) \rightarrow ((x:Var) \land (\text{typenv } x) = A \rightarrow (\text{type_body } S (c x) B)).

provided the maps \text{stack} and \text{typenv} are consistent w.r.t. the store-type \(S\). The proof of this property requires the use of both \text{unsat} and the decidability of equality over variables. On the other hand, the proof of the lemmata \text{rename_term} and \text{rename_body}, previously displayed, requires the application of the \(\beta\text{-expansion}\) and extensionality.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have illustrated the benefits of using Natural Deduction in combination with Higher-Order Abstract Syntax and the Theory of Contexts for reasoning on object-calculi with binders in type theory-based logical frameworks. We have carried out our experiment on Abadi and Cardelli’s impc, an object-based calculus featuring types and side-effects. Natural Deduction style of proof has allowed to distribute the hypotheses of proof derivations both the reduction and typing context of impc, thus obtaining judgments and proofs appreciably simpler than traditional ones. Weak HOAS has permitted to deal with the binders of impc in the Calculus of Inductive Constructions without having to encode neither \(\alpha\)-equivalence (which is inherited from the metalanguage) nor the substitution (which is not required by the calculus). This is a big advantage from the point of view of computer aided formal reasoning w.r.t. first-order techniques, as de Bruijn indexes or explicit names. A consequence of our choices is that we have obtained a more fine-grained treatment of closures. Finally, for reasoning on the formalization of impc in CIC we have adopted the Theory of Contexts, in order to gain the extra power CIC needs for reasoning on HOAS encodings.

Thus we have obtained a clean and compact formalization of impc in the proof assistant Coq, the implementation of CIC. Our style of encoding has allowed to prove formally a Subject Reduction theorem, which is particularly involved already “on paper”, with relatively little effort. Just notice that the full proof development amounts approximately to 112K byte and the size of the .vo file is 785K byte, working with Coq V7.3 on a Sun UltraSPARC Ile (64 bit).

To our knowledge, this is the first development of the theory of an object-based language with side effects, in LF based on type theory. The closest work may be [21], where Abadi and Cardelli’s functional object-based calculus \(ObL<\nu>\) is encoded in Coq, using traditional first-order techniques and Natural Semantics specifications through the Centaur system. A logic for reasoning on paper about object-oriented programs with imperative semantics, aliasing and self-reference in objects, has been presented in [1].

The Theory of Contexts has been already used with the weak HOAS for carrying out many case studies in recent years [see, e.g., [18, 19, 23, 29]]; however, the present work is the first application of the ToC to an object calculus. Our experience leads us to affirm that this approach is particularly well-suited with respect to the proof practice of Coq. The ToC can be actually plugged in existing LFs without requiring any redesign of these systems. In particular, it seems very suited for dealing with implementation-level structures, such as closures, since full substitution is not required.
Future work. As a first step, we plan to experiment further with the formalization we have carried out so far. We will consider other interesting (meta)properties of imp, beside the albeit fundamental Subject Reduction theorem. In particular, we can use the formalization for proving observational and behavioral equivalences of object programs.

From a practical point of view, our formalization could be used for the development of certified tools, such as interpreters, compilers and type checkers, for imp-like calculi and languages. We plan to certify a given tool with respect to the specification of the formal semantics of the object calculus and the target machine. We are confident that the use of Natural Deduction and HOAS should simplify these advanced tasks in the case of languages with binders.

From a more theoretical point of view, we had to modify slightly the unsaturation axiom of the ToC. This has occurred also in other applications of ToC [23, 29]. From all these case studies, we observe that these adaptments are required when the proof contexts have to satisfy some kind of "well-formedness" about variables: every time a new variable is generated, it has to come together with a set of other assumptions and informations (about its type, its kind, etc.). This points out that the current ToC can be generalized to subsume all these small variants, and that there is a connection with Schürmann’s regular world assumption [30], which should be investigated further.

6. REFERENCES


