An Open Challenge Problem Repository for Systems Supporting Binders

Amy Felty
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Canada
afelty@eecs.uottawa.ca

Alberto Momigliano
Dipartimento di Informatica
Università degli Studi di Milano
Milano, Italy
momigliano@di.unimi.it

Brigitte Pientka
School of Computer Science
McGill University
Montreal, Canada
bpientka@cs.mcgill.ca

A variety of logical frameworks support the use of higher-order abstract syntax in representing formal systems; however, each system has its own set of benchmarks. Even worse, general proof assistants that provide special libraries for dealing with binders offer a very limited evaluation of such libraries, and the examples given often do not exercise and stress-test key aspects that arise in the presence of binders. In this paper we design an open repository *ORBI* (Open challenge problem Repository for systems supporting reasoning with BInders). We believe the field of reasoning about languages with binders has matured, and a common set of benchmarks provides an important basis for evaluation and qualitative comparison of different systems and libraries that support binders, and it will help to advance the field.

1 Introduction

A variety of logical frameworks support the use of higher-order abstract syntax (HOAS) in representing formal systems; however, each system has its own set of benchmarks, often encoding the same object logics with minor differences. Even worse, general proof assistants that provide special libraries for dealing with binders often offer only a very limited evaluation of such libraries, and the examples given often do not exercise and stress-test key aspects that arise in the presence of binders.

The POPLMARK challenge [2] was an important milestone in surveying the state of the art in mechanizing the meta-theory of programming languages. We ourselves proposed several specific benchmarks [8] that are *crafted* to highlight the differences between the designs of various meta-languages with respect to reasoning with and within a context of assumptions, and we compared their implementation in four systems: the logical framework Twelf [23], the dependently-typed functional language Beluga [17, 18], the two-level Hybrid system [6, 15] as implemented on top of Coq and Isabelle/HOL, and the Abella system [10]. Finally, several systems that support reasoning with binders, in particular systems concentrating on modeling binders using HOAS, also provide a large collection of examples and case studies. For example, Twelf's wiki (`http://twelf.org/wiki/Case_studies`), Abella's library (`http://abella-prover.org/examples`), Beluga's distribution, and the Coq implementation of Hybrid (`http://www.site.uottawa.ca/~afelty/HybridCoq/`) contain sets of examples that highlight the many issues surrounding binders.

As the field matures, we believe it is important to be able to systematically and qualitatively evaluate approaches that support reasoning with binders. Having benchmarks is a first step in this direction. In this paper, we propose a common infrastructure for representing challenge problems and a central, Open challenge problem Repository for systems supporting reasoning with BInders (ORBI) for sharing benchmark problems based on the notation we have developed.

I. Cervesato and K. Chaudhuri (Eds.): Tenth International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice
EPTCS 185, 2015, pp. 18–32, doi:10.4204/EPTCS.185.2

ORBI is designed to be a human-readable, easily machine-parsable, uniform, yet flexible and extensible language for writing specifications of formal systems including grammar, inference rules, contexts and theorems. The language directly upholds HOAS representations and is oriented to support the mechanization of benchmark problems in Twelf, Beluga, Abella, and Hybrid, without hopefully precluding other existing or future HOAS systems. At the same time, we hope it also is amenable to translations to systems using other representation techniques such as nominal ones.

We structure the language in two parts:

1. the problem description, which includes the grammar of the object language syntax, inference rules, context schemas, and context relations
2. the logic language, which includes syntax for expressing theorems and directives to ORBI2X[1] tools.

We begin in Sect. 2 with a running example. We consider the untyped lambda-calculus as an object logic (OL), and present the syntax, some judgments, and sample theorems. In Sect. 3, we present ORBI by giving its grammar and explaining how it is used to encode our running example; Sect. 3.1 and Sect. 3.2 present the two parts of this specification as discussed above. We discuss related work in Sect. 4.

We consider the notation that we present here as a first attempt at defining ORBI (Version 0.1), where the goal is to cover the benchmarks considered in [8]. As new benchmarks are added, we are well aware that we will need to improve the syntax and increase the expressive power—we discuss limitations and some possible extensions in Sect. 5.

2 A Running Example

The first question that we face when defining an OL is how to describe well-formed objects. Consider the untyped lambda-calculus, defined by the following grammar:

$$M ::= x \mid \mathsf{lam}\, x.M \mid \mathsf{app}\, M_1\, M_2.$$

To capture additional information that is often useful in proofs, such as when a given term is *closed*, it is customary to give inference rules in natural deduction style for well-formed terms using hypothetical and parametric judgments. However, it is often convenient to present hypothetical judgments in a *localized* form, reducing some of the ambiguity of the two-dimensional natural deduction notation, and providing more structure. We therefore introduce an *explicit* context for bookkeeping, since when establishing properties about a given system, it allows us to consider the variable case(s) separately and to state clearly when considering closed objects, i.e., an object in the empty context. More importantly, while structural properties of contexts are implicitly present in the natural deduction presentation of inference rules (where assumptions are managed informally), the explicit context presentation makes them more apparent and highlights their use in reasoning about contexts.

$$\frac{\mathsf{is\_tm}\ x \in \Gamma}{\Gamma \vdash \mathsf{is\_tm}\ x}\ tm_v \qquad \frac{\Gamma, \mathsf{is\_tm}\ x \vdash \mathsf{is\_tm}\ M}{\Gamma \vdash \mathsf{is\_tm}\ (\mathsf{lam}\, x.M)}\ tm_l \qquad \frac{\Gamma \vdash \mathsf{is\_tm}\ M_1 \qquad \Gamma \vdash \mathsf{is\_tm}\ M_2}{\Gamma \vdash \mathsf{is\_tm}\ (\mathsf{app}\, M_1\, M_2)}\ tm_a$$

[1] Following TPTP's nomenclature [25], we call "ORBI2X" any tool taking an ORBI specification as input; for example, the translator for Hybrid described in [13] turns syntax, inference rules, and context definitions of ORBI into input to the Coq version of Hybrid, and it is designed so that it can be adapted fairly directly to output Abella scripts.

Traditionally, a context of assumptions is characterized as a sequence of formulas $A_1, A_2, \ldots, A_n$ listing its elements separated by commas [12, 19]. In [7], we argue that this is not expressive enough to capture the structure present in contexts, especially when mechanizing object logics, and we define context *schemas* to introduce the required extra structure:

$$\begin{array}{rrcl} \text{Atom} & A & & \\ \text{Block of declarations} & D & ::= & A \mid D;A \\ \text{Context} & \Gamma & ::= & \cdot \mid \Gamma, D \\ \text{Schema} & S & ::= & D_s \mid D_s + S \end{array}$$

A context is a sequence of declarations D where a declaration is a block of individual atomic assumptions separated by ';'.[2] The ';' binds tighter than ','. We treat contexts as ordered, i.e., later assumptions in the context may depend on earlier ones, but not vice versa—this in contrast to viewing contexts as multi-sets. Just as types classify terms, a *schema* will classify meaningful structured sequences. A schema consists of declarations D_s, where we use the subscript s to indicate that the declaration occurring in a concrete context having schema S may be an *instance* of D_s. We use $+$ to denote the alternatives in a context schema. For well-formed terms, contexts have a simple structure where each block contains a single atom, expressed as the following schema declaration:

$$S_x := \mathsf{is\_tm}\ x.$$

We write Φ_x to represent a context *satisfying* schema S_x (and similarly for other context schemas appearing in this paper). Informally, this means that Φ_x has the form $\mathsf{is\_tm}\ x_1, \ldots, \mathsf{is\_tm}\ x_n$ where $n \geq 0$ and $x_1, \ldots, x_n$ are distinct variables. (See [7] for a more formal account.)

For our running example, we consider two more simple judgments. The first is *algorithmic equality* for the untyped lambda-calculus, written $(\mathsf{aeq}\ M\ N)$. We say that two terms are algorithmically equal provided they have the same structure with respect to the constructors.

$$\frac{\mathsf{aeq}\ x\ x \in \Gamma}{\Gamma \vdash \mathsf{aeq}\ x\ x}\ ae_v \qquad \frac{\Gamma, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M\ N}{\Gamma \vdash \mathsf{aeq}\ (\mathsf{lam}\, x.M)\ (\mathsf{lam}\, x.N)}\ ae_l \qquad \frac{\Gamma \vdash \mathsf{aeq}\ M_1\ N_1 \quad \Gamma \vdash \mathsf{aeq}\ M_2\ N_2}{\Gamma \vdash \mathsf{aeq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}\ ae_a$$

The second is *declarative equality* written $(\mathsf{deq}\ M\ N)$, which includes versions of the above three rules called de_v, de_l, and de_a, where aeq is replaced by deq everywhere, plus reflexivity, symmetry and transitivity shown below.[3]

$$\frac{}{\Gamma \vdash \mathsf{deq}\ M\ M}\ de_r \qquad \frac{\Gamma \vdash \mathsf{deq}\ N\ M}{\Gamma \vdash \mathsf{deq}\ M\ N}\ de_s \qquad \frac{\Gamma \vdash \mathsf{deq}\ M\ L \quad \Gamma \vdash \mathsf{deq}\ L\ N}{\Gamma \vdash \mathsf{deq}\ M\ N}\ de_t$$

These judgments give rise to the following schema declarations:

$$\begin{array}{rcl} S_{xa} & := & \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \\ S_{xd} & := & \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x \\ S_{da} & := & \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x; \mathsf{aeq}\ x\ x \end{array}$$

[2] This is an oversimplification, since there are well-known specifications where contexts have more structure, see the solution to the POPLMARK challenge in [16] and the examples in [26]. In fact, those are already legal ORBI specs.

[3] We acknowledge that this definition of declarative equality has a degree of redundancy: the assumption $\mathsf{deq}\ x\ x$ in rule de_l is not needed, since rule de_r plays the variable role. However, this formulation exhibits issues, such as *context subsumption*, that would otherwise require more complex benchmarks.

The first two come directly from the ae_l and de_l rules where declaration blocks come in pairs. The third combines the two, and is used below in stating one of the example theorems.

When stating properties, we often need to relate two judgments to each other, where each one has its own context. For example, we may want to prove statements such as "if $\Phi_x \vdash J_1$ then $\Phi_{xa} \vdash J_2$." The proofs in [8] use two approaches.[4] In the first, the statement is reinterpreted in the *smallest* context that collects all relevant assumptions; we call this the *generalized context* approach (G). The above statement becomes "if $\Phi_{xa} \vdash J_1$ then $\Phi_{xa} \vdash J_2$." As an example theorem, we consider the completeness of declarative equality with respect to algorithmic equality, of which we only show the interesting left-to-right direction.

Theorem 2.1 (Completeness, G Version)

Admissibility of Reflexivity *If* $\Phi_{xa} \vdash \mathsf{is\_tm}\ M$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ M$.

Admissibility of Symmetry *If* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ N\ M$.

Admissibility of Transitivity *If* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$ *and* $\Phi_{xa} \vdash \mathsf{aeq}\ N\ L$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ L$.

Main Theorem *If* $\Phi_{da} \vdash \mathsf{deq}\ M\ N$ *then* $\Phi_{da} \vdash \mathsf{aeq}\ M\ N$.

In the second approach, we state how two (or more) contexts are related via context relations. For example, the following relation captures the fact that $\mathsf{is\_tm}\ x$ will occur in Φ_x in sync with an assumption block containing $\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x$ in Φ_{xa}.

$$\frac{}{\cdot \sim \cdot} \qquad \frac{\Phi_x \sim \Phi_{xa}}{\Phi_x, \mathsf{is\_tm}\ x \sim \Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x}$$

Similarly, we can define $\Phi_{xa} \sim \Phi_{xd}$.

$$\frac{}{\cdot \sim \cdot} \qquad \frac{\Phi_{xa} \sim \Phi_{xd}}{\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \sim \Phi_{xd}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x}$$

We call this the *context relations* approach (R). The theorems are then typically stated as: "if $\Phi_x \vdash J_1$ and $\Phi_x \sim \Phi_{xa}$ then $\Phi_{xa} \vdash J_2$." We can then revisit the completeness theorem for algorithmic equality together with the necessary lemmas as follows.

Theorem 2.2 (Completeness, R Version)

Admissibility of Reflexivity *Assume* $\Phi_x \sim \Phi_{xa}$. *If* $\Phi_x \vdash \mathsf{is\_tm}\ M$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ M$.

Admissibility of Symmetry *If* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ N\ M$.

Admissibility of Transitivity *If* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$ *and* $\Phi_{xa} \vdash \mathsf{aeq}\ N\ L$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ L$.

Main Theorem *Assume* $\Phi_{xa} \sim \Phi_{xd}$. *If* $\Phi_{xd} \vdash \mathsf{deq}\ M\ N$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$.

3 ORBI

ORBI aims to provide a common framework for systems that support reasoning with binders. Currently, our design is geared towards systems supporting HOAS, where there are (currently) two main approaches. On one side of the spectrum we have systems that implement various dependently-typed

[4] In proofs on paper, the differences between the two approaches usually do not appear; they are present in the details that are left implicit, but must be made explicit when mechanizing proofs. For example, on-paper versions of the admissibility of reflexivity that make these distinctions explicit appear in [7] as proofs of Theorems 7 and 8.

calculi. Such systems include Twelf, Beluga, and Delphin [20]. All these systems also provide, to various degrees, built-in support for reasoning modulo structural properties of a context of assumptions. These systems support inductive reasoning over terms as well as rules. Often it is more elegant in these systems to state theorems using the G-version [8].

On the other side there are systems based on a proof-theoretic foundation, which typically follow a two-level approach: they implement a specification logic (SL) inside a higher-order logic or type theory. Hypothetical judgments of object languages are modeled using implication in the SL and parametric judgments are handled via (generic) universal quantification. Contexts are commonly represented explicitly as lists or sets in the SL, and structural properties are established separately as lemmas. For example substituting for an assumption is justified by appealing to the cut-admissibility lemma of the SL. These lemmas are not directly and intrinsically supported through the SL, but may be integrated into a system's automated proving procedures, usually via tactics. Induction is usually only supported on derivations, but not on terms. Systems following this philosophy include Hybrid and Abella. Often these systems are better suited to proving R-versions of theorems.

The desire for ORBI to cater to both type and proof theoretic frameworks requires an almost impossible balancing act between the two views. For example, contexts are first-class and part of the specification language in Beluga; in Twelf, schemas for contexts are part of the specification language, which is an extension of LF, but users cannot explicitly quantify over contexts and manipulate them as first-class objects; in Abella and Hybrid, contexts are (pre)defined using inductive definitions on the reasoning level. We will describe next our common infrastructure design, directives, and guidelines that allow us to cater to existing systems supporting HOAS.

3.1 Problem Description in ORBI

ORBI's language for defining the grammar of an object language together with inference rules is based on the logical framework LF; pragmatically, we have adopted the concrete syntax of LF specifications in Beluga, which is almost identical to Twelf's. The advantage is that specifications can be directly parsed and more importantly type checked by Beluga, thereby eliminating many syntactically correct but meaningless expressions.

Object languages are written according to the EBNF grammar in Fig. 1, which uses certain standard conventions: `{a}` means repeat a production zero or more times, and comments in the grammar are enclosed between `(*` and `*)`. The token `id` refers to identifiers starting with a lower or upper case letter. These grammar rules are basically the standard ones used both in Twelf and Beluga and we do not discuss them in detail here. We only note that while the presented grammar permits general dependent types up to level *n*, ORBI specifications will only use level 0 and level 1. Intuitively, specifications at level 0 define the syntax of a given object language, while specifications at level 1 (i.e., type families that are indexed by terms of level 0) describe the judgments and rules for a given OL. We exemplify the grammar relative to the example of algorithmic vs. declarative equality. For more example specifications, we refer the reader to our survey paper [8] or to `https://github.com/pientka/ORBI`.[5]

Syntax An ORBI file starts in the `Syntax` section with the declaration of the constants used to encode the syntax of the OL in question, here untyped lambda-terms, which are introduced with the declarations:

[5] The observant reader will have noticed that ORBI's concrete syntax for schemas differs from the one that we have presented in Sect. 2, in so much that blocks are separated by commas and not by semi-colons. This is forced on us by our choice to re-use Beluga's parsing and checking tools.

```
sig      ::= {decl                        (* declaration *)
           | s_decl}                      (* schema declaration *)

decl     ::= id ":" tp "."                (* constant declaration *)
           | id ":" kind "."              (* type declaration *)

op_arrow ::= "->" | "<-"                  (* A <- B same as B -> A *)

kind     ::= type
           | tp op_arrow kind             (* A -> K *)
           | "{" id ":" tp "}" kind       (* Pi x:A.K *)

tp       ::= id {term}                    (* a M1 ... M2 *)
           | tp op_arrow tp
           | "{" id ":" tp "}" tp         (* Pi x:A.B *)

term     ::= id                           (* constants, variables *)
           | "\" id "." term              (* lambda x. M *)
           | term term                    (* M N *)

s_decl    ::= schema s_id "=" alt_blk ";"

s_id      ::= id

alt_blk   ::= blk {"+" blk}

blk       ::= block id ":" tp {"," id ":" tp}
```

Figure 1: ORBI grammar for syntax, judgments, inference rules, and context schemas

```
%% Syntax
tm: type.
app: tm -> tm -> tm.
lam: (tm -> tm) -> tm.
```

The declaration introducing type `tm` along with those of the constructors `app` and `lam` fully specify the syntax of OL terms. We represent binders in the OL using binders in the HOAS meta-language. Hence the constructor `lam` takes in a function of type `tm -> tm`. For example, the OL term $(\mathsf{lam}\, x.\, \mathsf{lam}\, y.\, \mathsf{app}\, x\, y)$ is represented as `lam (\x. lam (\y. app x y))`, where "\" is the binder of the metalanguage. Bound variables found in the object language are not explicitly represented in the meta-language.

Judgments and Rules These are introduced as LF type families (predicates) in the `Judgments` section followed by object-level inference rules for these judgments in the `Rules` section. In our running example, we have two judgments:

```
%% Judgments
aeq: tm -> tm -> type.
```

```
deq: tm -> tm -> type.
```

Consider first the inference rule for algorithmic equality for application, where the ORBI text is a straightforward encoding of the rule:

```
ae_a: aeq M1 N1 -> aeq M2 N2
   -> aeq (app M1 M2) (app N1 N2).
```

$$\frac{\Gamma \vdash \mathsf{aeq}\ M_1\ N_1 \quad \Gamma \vdash \mathsf{aeq}\ M_2\ N_2}{\Gamma \vdash \mathsf{aeq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}\ ae_a$$

Uppercase letters such as `M1` denote schematic variables, which are implicitly quantified at the outermost level, namely `{M1:tm}`, as is commonly done for readability purposes in Twelf and Beluga.
The binder case is more interesting:

```
ae_l: ({x:tm} aeq x x -> aeq (M x) (N x))
   -> aeq (lam (\x. M x)) (lam (\x. N x)).
```

$$\frac{\Gamma, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M\ N}{\Gamma \vdash \mathsf{aeq}\ (\mathsf{lam} x.M)\ (\mathsf{lam} x.N)}\ ae_l$$

We view the is_tm x assumption as the parametric assumption `x:tm`, while the hypothesis aeq x x (and its scoping) is encoded within the embedded implication `aeq x x -> aeq (M x) (N x)` in the current (informal) signature augmented with the dynamic declaration for `x`. As is well known, parametric assumptions and embedded implication are unified in the type-theoretic view. Note that the "variable" case, namely rule ae_v, is folded inside the binder case. We list here the rest of the `Rules` section:

```
%% Rules
de_a: deq M1 N1 -> deq M2 N2 -> deq (app M1 M2) (app N1 N2).
de_l: ({x:tm} deq x x -> deq (M x) (N x))
        -> deq (lam (\x. M x)) (lam (\x. N x)).
de_r: deq M M.
de_s: deq N M -> deq M N.
de_t: deq M L -> deq L N -> deq M N.
```

Schemas A schema declaration `s_decl` is introduced using the keyword `schema`. A `blk` consists of one or more declarations and `alt_blk` describes *alternating* schemas. For example, schemas mentioned in Sect. 2 appear in the `Schemas` section as:

```
%% Schemas
schema xG =  block (x:tm);
schema xaG = block (x:tm, u:aeq x x);
schema xdG = block (x:tm, u:deq x x);
schema daG = block (x:tm, u:deq x x, v:aeq x x);
```

To illustrate alternatives in contexts, consider extending our OL to the polymorphically typed lambda-calculus, which includes a new type `tp` in the `Syntax` section, and a new judgment:

```
atp: tp -> tp -> type.
```

representing equality of types in the `Judgments` section (as well as type constructors and rules for well-formed types and type equality, omitted here). With this extension, the following two examples replace the first two schemas in the `Schemas` section.

```
schema xG  = block (x:tm) + block (a:tp);
schema xaG = block (x:tm, u:aeq x x) + block (a:tp, v:atp a a);
```

While we type-check the schema definitions using an extension of the LF type checker (as implemented in Beluga), we do not verify that the given schema definition is meaningful with respect to the specification of the syntax and inference rules; in other words, we do not perform "world checking" in Twelf lingo.

Definitions So far we have considered the specification language for encoding formal systems. ORBI also supports declaring inductive definitions for specifying context relations. We start with the grammar for inductive definitions (Fig. 2). Although we plan to provide syntax for specifying more general inductive definitions, in this version of ORBI we *only* define *context relations* inductively, that is *n*-ary predicates between contexts of some given schemas. Hence the base predicate is of the form `id {ctx}` relating different contexts. For example, the `Definitions` section defines the relations $\Phi_x \sim \Phi_{xa}$ and

```
def_dec  ::= "inductive" id ":" r_kind "=" def_body ";"

r_kind   ::= "prop"
           | "{" id ":" s_id "}" r_kind

def_body ::= "|" id ":" def_prp {def_body}

def_prp  ::= id {ctx}
           | def_prp "->" def_prp

ctx      ::= [] | [id] | ctx "," id ":" blk
```

Figure 2: ORBI grammar for inductive definitions describing context relations

$\Phi_{xa} \sim \Phi_{xd}$. To illustrate, only the former is shown below.

```
%% Definitions
inductive Rxa : {g:xG} {h:xaG} prop =
| Rxa_nl: Rxa [] []
| Rxa_cs: Rxa [g] [h]
          -> Rxa [g, b:block (x:tm)] [h, b: block (x:tm, u:aeq x x)];
```

This kind of relation can be translated fairly directly to inductive n-ary predicates in systems supporting the proof-theoretic view. In the type-theoretic framework underlying Beluga, inductive predicates relating contexts correspond to recursive data types indexed by contexts; in fact ORBI adopts Beluga's concrete syntax, so as to directly type-check those definitions as well. Twelf's type theoretic framework, however, is not rich enough to support inductive definitions.

3.2 Theorems and Directives in ORBI

While the elements of an ORBI specification detailed in the previous subsection were relatively easy to define in a manner that is well understood by all the different systems we are targeting, we illustrate in this subsection those elements that are harder to describe uniformly due to the different treatment and meaning of contexts in the different systems.

Theorems We list the grammar for theorems in Fig. 3. Our reasoning language includes a category `prp` that specifies the logical formulas we support. The base predicates include `false`,`true`, term equality, atomic predicates of the form `id {ctx}`, which are used to express context relations, and predicates of the form `[ctx |- J]`, which represent judgments of an object language within a given context. Connectives and quantifiers include implication, conjunction, disjunction, universal and existential quantification over terms, and universal quantification over context variables.

```
thm      ::= "theorem" id ":" prp ";"

prp      ::= id {ctx}                      (* Context relation *)
           | "[" ctx  "|-" id {term} "]"   (* Judgment in a context *)
           | term "=" term                 (* Term equality *)
           | false                         (* Falsehood *)
           | true                          (* Truth *)
           | prp "&" prp                   (* Conjunction *)
           | prp "||" prp                  (* Disjunction *)
           | prp "->" prp                  (* Implication *)
           | quantif prp                   (* Quantification *)

quantif  ::= "{" id ":" s_id "}"           (* universal over contexts *)
           | "{" id ":" tp "}"             (* universal over terms *)
           | "<" id ":" tp ">"             (* existential over terms *)
```

Figure 3: ORBI grammar for theorems

To illustrate, the reflexivity lemmas and completeness theorems for both the G and R versions as they appear in the `Theorems` section are shown below. These theorems are a straightforward encoding of those stated in Sect. 2.

```
%% Theorems
theorem reflG: {h:xaG}{M:tm} [h |- aeq M M];
theorem ceqG:  {g:daG}{M:tm}{N:tm} [g |- deq M N] -> [g |- aeq M N];

theorem reflR: {g:xG}{h:xaG}{M:tm}        Rxa [g] [h] -> [h |- aeq M M];
theorem ceqR:  {g:xdG}{h:xaG}{M:tm}{N:tm} Rda [g] [h] ->
                                               [g |- deq M N] -> [h |- aeq M N];
```

As mentioned, we do not type-check theorems; in particular, we do not define the meaning of `[ctx |- J]`, since several interpretations are possible. In Beluga, every judgment `J` must be meaningful within the given context `ctx`; in particular, *terms* occurring in the judgment `J` must be meaningful in `ctx`. As a consequence, both parametric and hypothetical assumptions relevant for establishing the proof of `J` must be contained in `ctx`. Instead of the local context view adopted in Beluga, Twelf has one global ambient context containing all relevant parametric and hypothetical assumptions. Systems based on proof-theory such as Hybrid and Abella distinguish between assumptions denoting eigenvariables (i.e., parametric assumptions), which live in a global ambient context and proof assumptions (i.e., hypothetical assumptions), which live in the context `ctx`. While users of different systems understand how to interpret `[ctx |- J]`, reconciling these different perspectives in ORBI is beyond the scope of

this paper. Thus for the time being, we view theorem statements in ORBI as a kind of *comment*, where it is up to the user of a particular system to determine how to translate them.

Directives In ORBI, *directives* are comments that help the ORBI2X tools to generate target representations of the ORBI specifications. The idea is reminiscent of what *Ott* [24] does to customize certain declarations, e.g., the representation of variables, to the different programming languages/proof assistants it supports. The grammar for directives is listed in Fig. 4.

```
dir       ::=  '%' sy_set what decl {"," decl} {dest} '.'
             | '%%' sepr '.'

sy_id     ::=  hy | ab | bel | tw

sy_set    ::= '[' sy_id {',' sy_id} ']'

what      ::=  wf | explicit | implicit

dest      ::= 'in' ctx | 'in' s_id | 'in' id

sepr      ::=  Syntax | Judgments | Rules | Schemas | Definitions
             | Directives | Theorems
```

Figure 4: ORBI grammar for directives

The `sepr` directives, such as `Syntax`, are simply means to structure ORBI specifications. Most of the other directives that we consider in this version of ORBI are dedicated to help the translations into proof-theoretical systems, although we also include some to facilitate the translation of theorems to Beluga. The set of directives is not intended to be complete and the meaning of directives is system-specific. The directives `wf` and `explicit` are concerned with the asymmetry in the proof-theoretic view between declarations that give typing information, e.g., `tm:type`, and those expressing judgments, e.g., `aeq:tm -> tm -> type`. In Abella and Hybrid, the former may need to be reified in a judgment, in order to show that judgments preserve the well-formedness of their constituents, as well as to provide induction on the structure of terms; yet, in order to keep proofs compact and modular, we want to minimize this reification and only include them where necessary. The `Directives` section of our sample specification includes, for example,

```
% [hy,ab] wf tm.
```

which refers to the first line of the `Syntax` section where `tm` is introduced, and indicates that we need a predicate (e.g., `is_tm`) to express well-formedness of terms of type `tm`. Formulas expressing the definition of this predicate are automatically generated from the declarations of the constructors `app` and `lam` with their types.

The keyword `explicit` indicates when such well-formedness predicates should be included in the translation of the declarations in the `Rules` section. For example, the following formulas both represent possible translations of the `ae_l` rule to proof-theoretic systems. We use Abella's concrete syntax to exemplify:

```
aeq (lam M) (lam N) :- pi x\ is_tm x => aeq x x => aeq (M x) (N x).
aeq (lam M) (lam N) :- pi x\ aeq x x => aeq (M x) (N x).
```

where the typing information is explicit in the first and implicit in the second. By default, we choose the latter, that is well-formed judgments are assumed to be *implicit*, and require a directive if the former is desired. Consider, for example, that we want to conclude that whenever a judgment is provable, the terms in it are well-formed, e.g., if `aeq M N` is provable, then so are `is_tm M` and `is_tm N`. Such a lemma is indeed provable in Abella and Hybrid from the *implicit* translation of the rules for `aeq`. Proving a similar lemma for the `deq` judgment, on the other hand, requires some strategically placed explicit well-formedness information. In particular, the two directives:

```
% [hy,ab] explicit (x : tm) in de_l.
% [hy,ab] explicit (M : tm) in de_r.
```

require the clauses `de_l` and `de_r` to be translated to the following formulas:

```
deq (lam M) (lam N) :- pi x\ is_tm x => deq x x => deq (M x) (N x).
deq M M :- is_tm M.
```

The case for schemas is analogous. In the systems based on proof-theoretic approaches, contexts are typically represented using lists and schemas are translated to unary inductive predicates that verify that these lists have a particular regular structure. We again leave typing information implicit in the translation unless a directive is included. For example, the `xaG` schema with no associated directive will be translated to the following inductive definition in Abella:

```
Define aG : olist -> prop by
  xaG nil;
  nabla x, xaG (aeq x x :: As) := xaG As.
```

The directive `% [hy,ab] explicit (x : tm) in daG` will yield this Hybrid definition:

```
Inductive daG : list atm -> Prop :=
| nil_da : daG nil
| cns_da : forall (Gamma:list atm) (x:uexp),
    proper x -> daG Gamma -> daG (is_tm x :: deq x x :: aeq x x :: Gamma).
```

Similarly, directives in context relations, such as:

```
% [hy,ab] explicit (x : tm) in g in Rxa.
```

also state which well-formedness annotations to make explicit in the translated version. In this case, when translating the definition of `Rxa` in the `Definitions` section, they are to be kept in `g`, but skipped in `h`.

Keeping in mind that we consider the notion of directive *open* to cover other benchmarks and different systems, we offer some speculation about directives that we may need to translate theorems for the examples and systems that we are considering. For example, theorem `reflG` is proven by induction over `M`. As a consequence, `M` must be explicit.

```
% [hy,ab,bel] explicit (M : tm) in h in reflG.
```

The ORBI2Hybrid and ORBI2Abella tools will interpret the directive by adding an explicit assumption, as illustrated by the result of the ORBI2Abella translation:

```
forall H M, xaG H -> {H |- is_tm M} -> {H |- aeq M M}.
```

In Beluga, the directive is interpreted as:

```
{h:xaG} {M:[h |- tm]} [h |- aeq M M].
```

where M will have type tm in the context h. Moreover, since the term M is used in the judgment aeq within the context h, we associate M with an identity substitution, which is not displayed. In short, the directive allows us to lift the type specified in ORBI to a contextual type that is meaningful in Beluga. In fact, Beluga always needs additional information on how to interpret terms—are they closed or can they depend on a given context? For translating symG for example, we use the following directive to indicate the dependence on the context:

```
% [bel] implicit (M : tm), (N : tm) in h in symG.
```

3.3 Guidelines

In addition, we introduce a set of *guidelines* for ORBI specification writers, with the goal of helping translators generate output that is more likely to be accepted by a specific system. ORBI 0.1 includes four such guidelines, which are motivated by the desire to avoid putting too many constraints in the grammar rules. First, as we have seen in our examples, we use as a convention that free variables which denote schematic variables in rules are written using upper case identifiers; we use lower case identifiers for eigenvariables in rules and for context variables. Second, while the grammar does not restrict what types we can quantify over, the intention is that we quantify over types of level-0, i.e., objects of the syntax level, only. Third, in order to more easily accommodate systems without dependent types, Pi should not be used when writing non-dependent types; an arrow should be used instead. (In LF, for example, A -> B is an abbreviation for Pi x:A.B for the case when x does not occur in B. Following this guideline means favoring this abbreviation whenever it applies.) Fourth, when writing a context (grammar ctx), distinct variable names should be used in different blocks.

4 Related Work

Our approach to structuring contexts of assumptions takes its inspiration from Martin-Löf's theory of judgments, especially in the way it has been realized in Edinburgh LF. However, our formulation owes more to Beluga's type theory, where contexts are first-class citizens, than to the notion of *regular world* in Twelf.

The creation and sharing of a library of benchmarks has proven to be very beneficial to the field it represents. The brightest example is *TPTP* [25], whose influence on the development, testing and evaluation of automated theorem provers cannot be underestimated. Clearly our ambitions are much more limited. We have also taken some inspiration from its higher-order extension *THF0* [3], in particular in its construction in stages.

The success of TPTP has spurred other benchmark suites in related subjects, see for example *SATLIB* [14]; however, the only one concerned with induction is the *Induction Challenge Problems* (http://www.cs.nott.ac.uk/~lad/research/challenges), a collection of examples geared to the *automation* of inductive proof. The benchmarks are taken from arithmetic, puzzles, functional programming specifications, etc. and as such have little connection with our endeavor. On the other hand, the examples mentioned earlier coming from Twelf's wiki, Abella's library, Beluga's distribution, and Hybrid's web page contain a set of examples that highlight the issues around binders. As such they are prime candidates to be included in ORBI.

Other projects have put forward LF as a common ground: the goal of *Logosphere*'s (http://www.logosphere.org) was the design of a representation language for logical formalisms, individual theories, and proofs, with an interface to other theorem proving systems that were somewhat connected, but

the project never materialized. *SASyLF* [1] originated as a tool to teach programming language theory: the user specifies the syntax, judgments, theorems *and* proofs thereof (albeit limited to *closed* objects) in a paper-and-pencil HOAS-friendly way and the system converts them to totality-checked Twelf code. The capability to express and share proofs is of obvious interest to us, although such proofs, being a literal proof verbalization of the corresponding Twelf type family, are irremediably verbose. Finally, work on modularity in LF specifications [21] is of critical interest to give more structure to ORBI files.

Why3 (`http://why3.lri.fr`) is a software verification platform that intends to provide a front-end to third-party theorem provers, from proof assistants such as Coq to SMT-solvers. To this end Why3 provides a first-order logic with rank-1 polymorphism, recursive definitions, algebraic data types and inductive predicates [9], whose specifications are then translated to the several systems that Why3 supports. Typically, those translations are forgetful, but sometimes, e.g., with respect to Coq, they add some annotations, for example to ensure non-emptiness of types. Although we are really not in the same business as Why3, there are several ideas that are relevant; to name one, the notion of a *driver*, that is, a configuration file to drive transformations specific to a system. Moreover, Why3 provides an API for users to write and implement their own drivers and transformations.

Ott [24] is a highly engineered tool for "working semanticists," allowing them to write programming language definitions in a style very close to paper-and-pen specifications; then those are compiled into LaTeX and, more interestingly, into proof assistant code, currently supporting Coq, Isabelle/HOL, and HOL. Ott's metalanguage is endowed with a rich theory of binders, but at the moment it favors the "concrete" (non α-quotiented) representation, while providing support for the nameless representation for a single binder. Conceptually, it would be natural to extend Ott to generate ORBI code, as a bridge for Ott to support HOAS-based systems. Conversely, an ORBI user would benefit from having Ott as a front-end, since the latter view of grammar and judgment seems at first sight general enough to support the notion of schema and context relation.

In the category of environments for programming language descriptions, we mention *PLT-Redex* [5] and also the *K* framework [22]. In both, several large-scale language descriptions have been specified and tested. However, none of those systems has any support for binders, let alone context specifications, nor can any meta-theory be formally verified.

Finally, there is a whole research area dedicated to the handling and sharing of mathematical content (*MMK* `http://www.mkm-ig.org`) and its representation (*OMDoc* `https://trac.omdoc.org/OMDoc`), which is only very loosely connected to our project.

5 Conclusion

We have presented the preliminary design of a language, and more generally, of a common infrastructure for representing challenge problems for HOAS-based logical frameworks. The common notation allows us to express the syntax of object languages that we wish to reason about, as well as the context schemas, the judgments and inference rules, and the statements of benchmark theorems.

We strongly believe that the field has matured enough to benefit from the availability of a set of benchmarks on which qualitative and hopefully quantitative comparison can be carried out. We hope that ORBI will foster sharing of examples in the community and provide a common set of examples. We also see our benchmark repository as a place to collect and propose "open" challenge problems to push the development of meta-reasoning systems.

The challenge problems also play a role in allowing us, as designers and developers of logical frameworks, to highlight and explain how the design decisions for each individual system lead to differences

in using them in practice. Additionally, our benchmarks aim to provide a better understanding of what practitioners should be looking for, as well as help them foresee what kind of problems can be solved elegantly and easily in a given system, and more importantly, why this is the case. Therefore the challenge problems provide guidance for users and developers in better comprehending differences and limitations. Finally, they serve as an excellent regression suite.

The description of ORBI presented here is best thought of as a stepping stone towards a more comprehensive specification language, much as *THF0* [3] has been extended to the more expressive formalism THF_i, adding for instance, rank-1 polymorphism. Many are the features that we plan to provide in the near future, starting from general (monotone) *(co)inductive* definitions; currently we only relate contexts, while it is clearly desirable to relate arbitrary well-typed terms, as shown for example in [4] and [11] with respect to normalization proofs. Further, it is only natural to support infinite objects and behavior. However, full support for (co)induction is a complex matter, as it essentially entails fully understanding the relationship between the proof-theory behind Abella and Hybrid and the type theory of Beluga. Once this is in place, we can "rescue" ORBI theorems from their current status as comments and even include proof sketches in ORBI.

Clearly, there is a significant amount of implementation work ahead, mainly on the ORBI2X tools side, but also on the practicalities of the benchmark suite. Finally, we would like to open up the repository to other styles of formalization such as nominal, locally nameless, etc.

References

[1] Jonathan Aldrich, Robert J. Simmons & Key Shin (2008): *SASyLF: An Educational Proof Assistant for Language Theory*. In: *International Workshop on Functional and Declarative Programming in Education*, ACM Press, pp. 31–40, doi:10.1145/1411260.1411266.

[2] Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich & Steve Zdancewic (2005): *Mechanized Metatheory for the Masses: The* POPLMARK *Challenge*. In: *Eighteenth International Conference on Theorem Proving in Higher Order Logics, LNCS* 3603, Springer, pp. 50–65, doi:10.1007/11541868_4.

[3] Christoph Benzmüller, Florian Rabe & Geoff Sutcliffe (2008): *THF0—The Core of the TPTP Language for Higher-Order Logic*. In: *Fourth International Joint Conference on Automated Reasoning, LNCS* 5195, Springer, pp. 491–506, doi:10.1007/978-3-540-71070-7_41.

[4] Andrew Cave & Brigitte Pientka (2012): *Programming with Binders and Indexed Data-Types*. In: *Thirty-Ninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM Press, pp. 413–424, doi:10.1145/2103656.2103705.

[5] Matthias Felleisen, Robert Bruce Findler & Matthew Flatt (2009): *Semantics Engineering with PLT Redex*. The MIT Press.

[6] Amy P. Felty & Alberto Momigliano (2012): *Hybrid: A Definitional Two-Level Approach to Reasoning with Higher-Order Abstract Syntax*. *Journal of Automated Reasoning* 48(1), pp. 43–105, doi:10.1007/s10817-010-9194-x.

[7] Amy P. Felty, Alberto Momigliano & Brigitte Pientka (2015): *The Next 700 Challenge Problems for Reasoning with Higher-Order Abstract Syntax Representations: Part 1—A Common Infrastructure for Benchmarks*. *CoRR* abs/1503.06095. Available at http://arxiv.org/abs/1503.06095.

[8] Amy P. Felty, Alberto Momigliano & Brigitte Pientka (2015): *The Next 700 Challenge Problems for Reasoning with Higher-Order Abstract Syntax Representations: Part 2—A Survey*. *Journal of Automated Reasoning*. (to appear).

[9] Jean-Christophe Filliâtre (2013): *One Logic To Use Them All*. In: *Twenty-Fourth International Conference on Automated Deduction, LNCS* 7898, Springer, pp. 1–20, doi:10.1007/978-3-642-38574-2_1.

[10] Andrew Gacek (2008): *The Abella Interactive Theorem Prover (System Description)*. In: *Fourth International Joint Conference on Automated Reasoning, LNCS* 5195, Springer, pp. 154–161, doi:10.1007/978-3-540-71070-7_13.

[11] Andrew Gacek, Dale Miller & Gopalan Nadathur (2012): *A Two-Level Logic Approach to Reasoning About Computations*. *Journal of Automated Reasoning* 49(2), pp. 241–273, doi:10.1007/s10817-011-9218-1.

[12] J.-Y. Girard, Y. Lafont & P. Tayor (1990): *Proofs and Types*. Cambridge University Press.

[13] Nada Habli & Amy P. Felty (2013): *Translating Higher-Order Specifications to Coq Libraries Supporting Hybrid Proofs*. In: *Third International Workshop on Proof Exchange for Theorem Proving, EasyChair Proceedings in Computing* 14, pp. 67–76.

[14] Holger H. Hoos & Thomas Stützle (2000): *SATLIB: An Online Resource for Research on SAT*. In: *SAT 2000: Highlights of Satisfiability Research in the Year 2000, Frontiers in Artificial Intelligence and Applications* 63, IOS Press, pp. 283–292.

[15] Alberto Momigliano, Alan J. Martin & Amy P. Felty (2008): *Two-Level Hybrid: A System for Reasoning Using Higher-Order Abstract Syntax*. In: *Second International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP 2007, ENTCS* 196, Elsevier, pp. 85–93, doi:10.1016/j.entcs.2007.09.019.

[16] Brigitte Pientka (2007): *Proof Pearl: The Power of Higher-Order Encodings in the Logical Framework LF*. In: *Twentieth International Conference on Theorem Proving in Higher-Order Logics*, LNCS, Springer, pp. 246–261, doi:10.1007/978-3-540-74591-4_19.

[17] Brigitte Pientka & Andrew Cave (2015): *Inductive Beluga:Programming Proofs (System Description)*. In: *Twenty-Fifth International Conference on Automated Deduction*, Springer.

[18] Brigitte Pientka & Joshua Dunfield (2010): *Beluga: A Framework for Programming and Reasoning with Deductive Systems (System Description)*. In: *Fifth International Joint Conference on Automated Reasoning, LNCS* 6173, Springer, pp. 15–21, doi:10.1007/978-3-642-14203-1_2.

[19] Benjamin C. Pierce (2002): *Types and Programming Languages*. MIT Press.

[20] Adam Poswolsky & Carsten Schürmann (2009): *System Description: Delphin—A Functional Programming Language for Deductive Systems*. In: *Third International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2008), ENTCS* 228, Elsevier, pp. 113–120, doi:10.1016/j.entcs.2008.12.120.

[21] Florian Rabe & Carsten Schürmann (2009): *A Practical Module System for LF*. In: *Fourth International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, ACM Press, pp. 40–48, doi:10.1145/1577824.1577831.

[22] Grigore Roşu & Traian Florin Şerbănuţă (2010): *An Overview of the K Semantic Framework*. *Journal of Logic and Algebraic Programming* 79(6), pp. 397–434, doi:10.1016/j.jlap.2010.03.012.

[23] Carsten Schürmann (2009): *The Twelf Proof Assistant*. In: *Twenty-Second International Conference on Theorem Proving in Higher Order Logics, LNCS* 5674, Springer, pp. 79–83, doi:10.1007/978-3-642-03359-9_7.

[24] Peter Sewell, Francesco Zappa Nardelli, Scott Owens, Gilles Peskine, Thomas Ridge, Susmit Sarkar & Rok Strniša (2010): *Ott: Effective Tool Support for the Working Semanticist*. *Journal of Functional Programming* 20(1), pp. 71–122, doi:10.1017/S0956796809990293.

[25] Geoff Sutcliffe (2009): *The TPTP Problem Library and Associated Infrastructure*. *Journal of Automated Reasoning* 43(4), pp. 337–362, doi:10.1007/s10817-009-9143-8.

[26] Yuting Wang, Kaustuv Chaudhuri, Andrew Gacek & Gopalan Nadathur (2013): *Reasoning About Higher-Order Relational Specifications*. In: *Fifteenth International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, ACM Press, pp. 157–168, doi:10.1145/2505879.2505889.