Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

The Next 700 Challenge Problems for Reasoning with Higher-Order Abstract Syntax Representations

Part 1—A Common Infrastructure for Benchmarks

Amy P. Felty · Alberto Momigliano · Brigitte Pientka

November 27, 2014

Abstract A variety of logical frameworks support the use of higher-order abstract syntax (HOAS) in representing formal systems. Although these systems seem superficially the same, they differ in a variety of ways; for example, how they handle a *context* of assumptions and which theorems about a given formal system can be concisely expressed and proved. Our contributions in this paper are three-fold: 1) we develop a common infrastructure for representing benchmarks for systems supporting reasoning with binders, 2) we present several concrete benchmarks, which highlight a variety of different aspects of reasoning within a context of assumptions, and 3) we design an open repository *ORBI* (Open challenge problem Repository for systems supporting reasoning with BInders). Our work sets the stage for providing a basis for qualitative comparison of different systems. This allows us to review and survey the state of the art, which we do in great detail for four systems in Part 2 of this paper (Felty et al, 2014). It also allows us to outline future fundamental research questions regarding the design and implementation of meta-reasoning systems.

Keywords Logical Frameworks · Higher-Order Abstract Syntax · Context Reasoning · Benchmarks

1 Introduction

In recent years the PoplMark challenge (Aydemir et al, 2005) has stimulated considerable interest in mechanizing the meta-theory of programming languages

A. P. Felty
School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada, E-mail: afelty@eecs.uottawa.ca

A. Momigliano
Dipartimento di Informatica, Università degli Studi di Milano, Italy, E-mail: momigliano@di.unimi.it

B. Pientka
School of Computer Science, McGill University, Montreal, Canada, E-mail: bpientka@cs.mcgill.ca

and it has played a substantial role in the wide-spread use of proof assistants to prove properties, for example, of parts of a compiler or of a language design. The PoplMark challenge concentrated on summarizing the state of the art, identifying best practices for (programming language) researchers embarking on formalizing language definitions, and identifying a list of engineering improvements to make the use of proof assistants (more) common place. While these are important questions whose answers will foster the adoption of proof assistants by non-experts, it neglects some of the deeper fundamental questions: What should existing or future meta-languages and meta-reasoning environments look like and what requirements should they satisfy? What support should an ideal meta-language and proof environment give to facilitate mechanizing meta-reasoning? How can its design reflect and support these ideals?

We believe "good" meta-languages should free the user from dealing with tedious bureaucratic details, so s/he is able to concentrate on the essence of a proof or algorithm. Ultimately, this means that users will mechanize proofs more quickly. In addition, since effort is not wasted on cumbersome details, proofs are more likely to capture only the essential steps of the reasoning process, and as a result, may be easier to trust. For instance, weakening is a typical a low-level lemma that is used pervasively (and silently) in a proof. Freeing the user of such details ultimately may also mean that the automation of such proofs is more feasible.

One fundamental question when mechanizing formal systems and their meta-theory is how to represent variables and variable binding structures. There is a wide range of answers to this question from using de Bruijn indices to locally nameless representations, and nominal encodings, etc. For a partial view of the field see the papers collected in the *Journal of Automated Reasoning*'s special issue dedicated to PoplMark (Pierce and Weirich, 2012) and the one on "Abstraction, Substitution and Naming" (Fernández and Urban, 2012).

Encoding object languages and logics (OLs) via higher-order abstract syntax (HOAS), sometimes referred to as "lambda-tree syntax" (Miller and Palamidessi, 1999), where we utilize meta-level binders to model object-level binders is in our opinion the most advanced technology. HOAS avoids implementing common although notoriously tricky routines dealing with variables, such as capture-avoiding substitution, renaming, and fresh name generation. Compared to other techniques, HOAS leads to very concise and elegant encodings and provides significant support for such an endeavor. Concentrating on encoding binders, however, neglects another important and fundamental aspect: the support for hypothetical and parametric reasoning, in other words reasoning within a context of assumptions. Considering a derivation within a context is common place in programming language theory and leads to several natural questions: How do we model the context of assumptions? How do we know that a derivation is sensible within the scope of a context? Can we model the relationships between different contexts? How do we deal with structural properties of contexts such as weakening, strengthening, and exchange? How do we know assumptions in a context occur uniquely? How do we take advantage of the HOAS approach to substitution?

Even in systems supporting HOAS there is not a uniform answer to these questions. On one side of the spectrum we have systems that implement various dependently-typed calculi. Such systems include the logical framework Twelf (Schürmann, 2009), the dependently-typed functional language Beluga (Pientka, 2008; Pientka and Dunfield, 2010), and Delphin (Poswolsky and Schürmann, 2008).

All these systems also provide, in various degrees, built-in support for reasoning modulo structural properties of a context of assumptions.

On the other side there are systems based on a proof-theoretic foundation, which 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. Systems following this philosophy are for instance the two-level Hybrid system (Momigliano et al, 2008; Felty and Momigliano, 2012) as implemented on top of Coq and Isabelle/HOL, and the Abella system (Gacek, 2008).

This paper, together with Part 2 (Felty et al, 2014), is a major extension of an earlier conference paper (Felty and Pientka, 2010). The contributions of the present paper are three-fold. First, we develop a common framework and infrastructure for representing *benchmarks* for systems supporting reasoning with binders; in particular, we develop notation to view contexts as "structured sequences" and classify contexts using schemas. Moreover, we abstractly characterize in a uniform way basic structural properties that many object languages satisfy, such as weakening, strengthening, and exchange. This lays the foundation for describing benchmarks and comparing different approaches to mechanizing OLs. Second, we propose several challenge problems 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, in view of their mechanization in a given proof assistant. In Part 2 of this paper (Felty et al, 2014), we carry out such a comparison on four systems: Twelf, Beluga, Hybrid, and Abella. Third, we discuss the design of *ORBI* (Open challenge problem Repository for systems supporting reasoning with BInders), an open repository for sharing benchmark problems based on the infrastructure that we have developed. Although *ORBI*'s syntax is inspired by systems such as Twelf and Beluga, we do not commit to using a particular system, as we wish to retain the needed flexibility to be able to easily support translations to both type-theoretic and proof-theoretic approaches.[1] 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 the benchmark theorems. 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. This means reviewing the state of the art, as well as outlining future fundamental research questions regarding the design and implementation of meta-reasoning systems, as

[1] A first step in this direction is the translator for Hybrid, whose first version is presented in Habli and Felty (2013).

we discuss further in the companion paper (Felty et al, 2014). Additionally, our benchmarks aim to provide a better understanding of what practitioners should be looking for, as well as help them understand 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 understanding differences and limitations. Finally, they serve as an excellent regression suite.

This paper does not, of course, present 700 challenge problems. We start with a few and hope that others will contribute to the benchmark repository, implement these challenge problems, and further our understanding of the trade-offs involved in choosing one system over another for this kind of reasoning.

The paper is structured as follows: In Sect. 2 we motivate our definition of contexts as "structured sequences," which refines the standard view of contexts, and we describe generically and abstractly some context properties. Using this terminology we then present the benchmarks and their proofs in Sect. 3. In Sect. 4, we introduce ORBI and discuss how it provides HOAS encodings of the benchmarks in a uniform manner. We discuss related work in Sect. 5, before concluding in Sect. 6. Appendix A provides a quick reference guide to the benchmarks and Appendix B gives a complete example of an ORBI file for a selection of the benchmark problems. Full details about the challenge problems and their mechanization can be found at `https://github.com/pientka/ORBI`. The latter, as well as the present paper, can be better appreciated by reading the companion paper (Felty et al, 2014).

2 Contexts of Assumptions: Preliminaries and Terminology

Reasoning with and within a context of assumptions is common when we prove meta-theoretic properties about object languages such as type systems or logics. Hence, how to represent contexts and enforce properties such as well-formedness, weakening, strengthening, exchange, uniqueness of assumptions, and substitution is a central issue once we mechanize such reasoning.

As mentioned, proof environments supporting higher-order abstract syntax differ in how they represent and model contexts and our comparison (Felty et al, 2014) to a large extent focuses on this issue. Here we lay down a common framework and notation for describing the syntax of object languages, inference rules and contexts by using different representative examples. In particular, we refine the standard view of contexts as sequences of assumptions and abstractly describe structural properties such as weakening and exchange satisfied by many object languages. Our description follows mathematical practice, in contrast to giving a fully formal account based on, for example, type theory. In fact, all the notions that we touch upon in this section, such as substitution, α-renaming, bindings, context schemas to name a few, can and have been generally treated in Beluga (see e.g., Pientka, 2008). However, we deliberately choose to base our description on mathematical practice to make our benchmarks more accessible to a wider audience and so as not to force upon us one particular foundation. This infrastructure may be seen as a first step towards developing a formal translation between different foundations, i.e., a translation between Beluga's type-theoretic foundation and the proof-theory underlying systems such as Hybrid or Abella.

2.1 Defining Well-formed Objects

The first question that we face when defining an OL is how to describe well-formed objects. Consider the polymorphic lambda-calculus. Commonly the grammar of this language is defined using Backus-Naur form (BNF) as follows.

$$
\begin{array}{lll}
\text{Types } A, B & ::= & \alpha \mid \mathsf{arr} A\, B \mid \mathsf{all}\, \alpha.\, A \\
\text{Terms } M & ::= & x \mid \mathsf{lam}\, x.\, M \mid \mathsf{app}\ M_1\ M_2 \mid \mathsf{tlam}\, \alpha.\, M \mid \mathsf{tapp}\ M\ A
\end{array}
$$

The grammar, however, does not capture properties of interest such as when a given term or type is *closed*. Alternatively, we can describe well-formed types and terms as *judgments* using axioms and inference rules following Martin-Löf (1996), as popularized in programming language theory by Pfenning's *Computation and Deduction* notes (Pfenning, 2001).

We start with an *implicit-context* version of the rules for well-formed types and terms that plays the part of the above BNF grammar, but is also significantly more expressive. To describe whether a type A or term M is well-formed we use two judgments: $\boxed{\mathsf{is\_tp}\ A}$ and $\boxed{\mathsf{is\_tm}\ M}$, whose formation rules are depicted in Fig. 1. The rule for function types (tp_{ar}) is unsurprising. The rule tp_{al} states that a type $\mathsf{all}\, \alpha.\, A$ is well-formed if A is well-formed under the assumption that the variable α is also. We say that this rule is *parametric* in the name of the bound variable α—thus implicitly enforcing the usual eigenvariable condition, since bound variables can be α-renamed at will—and *hypothetical* in the name of the axiom (tp_v) stating the well-formedness of this type variable. In this two-dimensional representation, derived from Gentzen's presentation of natural deduction, we do not have an explicit rule for variables: instead, for each type variable introduced by tp_{al} we also introduce the well-formedness assumption about that variable, and we explicitly include names for the bound variable and axiom as parameters to the rule name.

$\boxed{\mathsf{is\_tp}\ A}$ – Type A is well-formed

$$
\dfrac{\begin{array}{c} \dfrac{}{\mathsf{is\_tp}\ \alpha}\ tp_v \\ \vdots \\ \mathsf{is\_tp}\ A \end{array}}{\mathsf{is\_tp}\ (\mathsf{all}\, \alpha.\, A)}\ tp_{al}^{\alpha, tp_v}
\qquad
\dfrac{\mathsf{is\_tp}\ A \quad \mathsf{is\_tp}\ B}{\mathsf{is\_tp}\ (\mathsf{arr} A\, B)}\ tp_{ar}
$$

$\boxed{\mathsf{is\_tm}\ M}$ – Term M is well-formed

$$
\dfrac{\begin{array}{c} \dfrac{}{\mathsf{is\_tm}\ x}\ tm_v \\ \vdots \\ \mathsf{is\_tm}\ M \end{array}}{\mathsf{is\_tm}\ (\mathsf{lam}\, x.\, M)}\ tm_{l}^{x, tm_v}
\qquad
\dfrac{\begin{array}{c} \dfrac{}{\mathsf{is\_tp}\ \alpha}\ tp_v \\ \vdots \\ \mathsf{is\_tm}\ M \end{array}}{\mathsf{is\_tm}\ (\mathsf{tlam}\, \alpha.\, M)}\ tm_{tl}^{\alpha, tp_v}
$$

$$
\dfrac{\mathsf{is\_tm}\ M_1 \quad \mathsf{is\_tm}\ M_2}{\mathsf{is\_tm}\ (\mathsf{app}\ M_1\ M_2)}\ tm_a
\qquad
\dfrac{\mathsf{is\_tm}\ M \quad \mathsf{is\_tp}\ A}{\mathsf{is\_tm}\ (\mathsf{tapp}\ M\ A)}\ tm_{ta}
$$

Fig. 1 Well-formed Types and Terms (implicit context)

While variables might occur free in a type given via the BNF grammar, the two-dimensional implicit-context formulation models more cleanly the *scope* of variables; e.g., a type $\mathsf{is\_tp}\ (\mathsf{all}\,\alpha.\,\mathsf{arr}\ \alpha\,\beta)$ is only meaningful in the context where we have the assumption $\mathsf{is\_tp}\ \beta$.

Following this judgmental view, we can also characterize well-formed terms: the rule for term application (tm_a) is straightforward and the rule for type application (tm_{ta}) simply refers to the previous judgment for well-formed types since types are embedded in terms. The rules for term abstraction (tm_l) and type abstraction (tm_{tl}) are again the most interesting. The rule tm_l is parametric in the variable x and hypothetical in the assumption $\mathsf{is\_tm}\ x$; similarly the rule tm_{tl} is parametric in the type variable α and hypothetical in the assumption $\mathsf{is\_tp}\ \alpha$.

We emphasize that mechanizations of a given object language can use either one of these two representations, the BNF grammar or the judgmental implicit context formulation. However, it is important to understand how to move between these representations and the trade-offs and consequences involved. For example, if we choose to support the BNF-style representation of object languages in a proof assistant, we might need to provide basic predicates that verify whether a given object is closed; further we may need to reason explicitly about the scope of variables. HOAS-style proof assistants typically adopt the judgmental view providing a uniform treatment for objects themselves (well-formedness rules) and other inference rules about them.

2.2 Context Definitions

Introducing the appropriate assumption about each variable is a general methodology that scales to OLs accommodating much more expressive assumptions. For example, when we specify typing rules, we introduce a typing assumption that keeps track of the fact that a given variable has a certain type. This approach can also result in compact and elegant proofs. Yet, it is often convenient to present hypothetical judgments in a *localized* form, reducing some of the ambiguity of the two-dimensional notation. 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 above 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.

To contrast representation using explicit contexts to implicit ones and to highlight the differences, we re-formulate the earlier rules for well-formed types and terms given in Fig. 1 using explicit contexts in Sect. 2.4. As another example of using explicit contexts, we give the standard typing rules for the polymorphic lambda-calculus (see Sect. 2.4). The reader might want to skip ahead to get an intuition of what explicit contexts are and how they are used in practice. In the rest of this section, we first introduce terminology for structuring such contexts, and then describe structural properties they (might) satisfy.

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 (Pierce, 2002; Girard et al, 1990). However, we argue that this is not expressive enough to capture the structure present in contexts, especially when mechanizing OLs. In fact, there are two limitations from that point of view.

First, simply stating that a context is a sequence of formulas does not characterize adequately and precisely what assumptions can occur in a context and in what order. For example, to characterize a well-formed type, we consider a type in a context Φ_α of type variables. To characterize a well-formed term, we must consider the term in a context $\Phi_{\alpha x}$ that may contain type variables α *and* term variables x.

$$\begin{array}{rl}\text{Context } \Phi_\alpha & ::= \cdot \mid \Phi_\alpha, \mathsf{is\_tp}\ \alpha \\ \Phi_{\alpha x} & ::= \cdot \mid \Phi_{\alpha x}, \mathsf{is\_tp}\ \alpha \mid \Phi_{\alpha x}, \mathsf{is\_tm}\ x\end{array}$$

As a consequence, we need to be able to state in our mechanization when a given context *satisfies* being a well-formed context Φ_α or $\Phi_{\alpha x}$. In other words, the grammar for Φ_α and $\Phi_{\alpha x}$ will give rise to a *schema*, which describes when a context is meaningful. Simply stating that a context is a sequence of assumptions does not allow us necessarily to distinguish between different contexts.

Second, forming new contexts by a comma does not capture enough structure. For example, consider the typing rule for lambda-abstraction that states that $\mathsf{lam}\, x.\, M$ has type $(\mathsf{arr}\ C\, B)$, if assuming that x is a term variable and x has type C, we can show that M has type B. Note that whenever we introduce assumptions $x{:}C$ (read as "term variable x has type C"), we at the same time introduce the additional assumption that x is a *new* term variable. This is indeed important, since from it we can derive the fact that every typing assumption is unique. Simply stating that the typing context is a list of assumptions $x{:}C$, as shown below in the first attempt, fails to capture that x is a term variable, distinct from all other term variables. In fact, it says nothing about x.

$$\text{Typing context (attempt 1) } \Phi ::= \cdot \mid \Phi, x{:}C$$

The second attempt below also fails, because the occurrences of the comma have two different meanings.

$$\text{Typing context (attempt 2) } \Phi ::= \cdot \mid \Phi, \mathsf{is\_tm}\ x, x{:}C$$

The comma between $\mathsf{is\_tm}\ x, x{:}C$ indicates that whenever we have an assumption $\mathsf{is\_tm}\ x$, we also have an assumption $x{:}C$. These assumptions come in pairs and form one *block* of assumptions. On the other hand, the comma between Φ and $\mathsf{is\_tm}\ x, x{:}C$ indicates that the context Φ is *extended* by the block containing assumptions $\mathsf{is\_tm}\ x$ and $x{:}C$.

Taking into account such blocks leads to the definition of contexts as *structured sequences*. A context is a sequence of declarations D where a declaration is a block of individual atomic assumptions separated by ';'. 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.

We thus introduce the following categories:

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

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.

We can declare the schemas corresponding to the previous contexts, seen as structured sequences, as follows:

$$\begin{array}{ll} S_\alpha & ::= \mathsf{is\_tp}\ \alpha \\ S_{\alpha x} & ::= \mathsf{is\_tp}\ \alpha + \mathsf{is\_tm}\ x \\ S_{\alpha t} & ::= \mathsf{is\_tp}\ \alpha + \mathsf{is\_tm}\ x; x{:}C \end{array}$$

We use the following notational convention for declarations and schemas: Lower case letters denote bound variables (eigenvariables), obeying the Barendregt variable convention; $\mathsf{EV}(D)$ will denote the set of eigenvariables occurring in D. Upper case letters are used for "schematic" variables. Therefore, we can always rename the x in the declaration $\mathsf{is\_tm}\ x; x{:}C$ and instantiate C. For example, the context $\mathsf{is\_tm}\ y; y{:}\ \mathsf{nat},\ \mathsf{is\_tp}\ \alpha,\ \mathsf{is\_tm}\ z; z{:}\ (\mathsf{arr}\ \alpha\, \alpha)$ fits the schema $S_{\alpha t}$.[2]

We say that a declaration D is *well-formed* if for every $x \in \mathsf{EV}(D)$ there is an atom in D (notation $A \in D$) denoting the well-formedness judgment for x, which we generically refer to as *is_wf* x, with the proviso that *is_wf* x precedes its use in D; the meta-notation *is_wf* will be instantiated by an appropriate atom such as $\mathsf{is\_tm}$ or $\mathsf{is\_tp}$. A schema is *well-formed* if and only if all its declarations are well-formed. For example, the schema $S_{\alpha t}$ is well-formed since the x in $x{:}C$ is declared by $\mathsf{is\_tm}\ x$ appearing earlier in the same declaration. We will assume in the following that all schemas are such.

More generally, we say that a concrete context Γ *has schema* S (Γ $\mathsf{has\_schema}$ S), if every declaration in Γ is an instance of some schema declaration D_s in S. By convention, when we write S_l to denote a context schema, Γ_l will denote a valid instance of S_l, namely such that Γ_l $\mathsf{has\_schema}$ S_l, where subscript l is used to denote the relationship between the schema and an instance of it.

Schema Satisfaction $\boxed{\Gamma\ \mathsf{has\_schema}\ S}$

$$\frac{}{\cdot\ \ \mathsf{has\_schema}\ S} \qquad \frac{\Gamma\ \mathsf{has\_schema}\ S \qquad D \in S \qquad \mathsf{EV}(D) \cap \mathsf{EV}(\Gamma) = \emptyset}{(\Gamma, D)\ \mathsf{has\_schema}\ S}$$

Block D of Declaration is valid $\boxed{D \in S}$

$$\frac{D \text{ instance of } D_s}{D \in D_s} \qquad \frac{D \text{ instance of } D_s}{D \in D_s + S} \qquad \frac{D \in S}{D \in D_s + S}$$

Note that if $D \in S$, then it is by definition well-formed. The premise $\mathsf{EV}(D) \cap \mathsf{EV}(\Gamma) = \emptyset$ requires eigenvariables in different blocks in a context satisfying the schema to be distinct from each other. This constraint will always be satisfied by contexts that appear in proofs of judgments using our inference rules—again, see

[2] Although a schema does not appear to have an explicit binder, all the eigenvariables and schematic variables occurring are considered bound. In ORBI (see Sect. 4) the **block** keyword delineates the scope of eigenvariables and we use the convention that schematic variables are written using upper case letters. Beluga's type theory provides a formal type-theoretic foundation for describing schemas where the scope of eigenvariables and schematic variables in a schema is enforced using Σ and Π-types.

for example the inference rules in Sect. 2.4. We remark that a given context can in principle inhabit different schemas; for example the context $\mathsf{is\_tp}\ \alpha_1, \mathsf{is\_tp}\ \alpha_2$ has schema S_α but also inhabits schemas $S_{\alpha x}$ and $S_{\alpha t}$.

Note that according to the given grammar for schemas, contexts contain only atomic assumptions. We could consider non-atomic assumptions; in fact, more complex assumptions are not only possible, but sometimes yield very compact and elegant specifications, as we touch upon in Sect. 6. However, to account for them, we would need to introduce a language for terms and formulas that we feel would detract from the goal at hand.

2.3 Structural Properties of Contexts

So far we have introduced terminology for describing objects in three different ways: using a BNF grammar, defining objects and rules via a two-dimensional implicit context, and using an explicit context containing structured sequences of assumptions following a given context schema. For the latter, we have not yet described the associated inference rules. Before we do (in Sect. 2.4 as mentioned), we introduce structural properties of explicit contexts *generically* and *abstractly*.

We concentrate here on developing a common framework for describing object languages including structural properties they might satisfy. However, we emphasize that whether a given object language does admit structural properties such as weakening or exchange is a property that needs to be verified on a case-by-case basis. In the subsequent discussion and in all our benchmarks, we concentrate on examples satisfying weakening, exchange, and strengthening, i.e., assumptions can be used as often as needed, they can be used in any order, and certain assumptions will be known not to be needed.

Our refined notion of context has an impact on structural properties of contexts: e.g., weakening can be described by adding a new declaration to a context, as well as adding an element inside a block of declarations. We distinguish between structural properties of a *concrete* context and structural properties of *all* contexts of a given schema. For example, given the context schemas S_α and $S_{\alpha x}$, we know that all concrete contexts of schema $S_{\alpha x}$ can be *strengthened* to obtain a concrete context of schema S_α. Dually, we can think of *weakening* a context of schema S_α to a context of schema $S_{\alpha x}$. We introduce the operations rm and perm, where rm removes an element of a declaration, and perm permutes the elements within a declaration.

Definition 1 (Operations on Declarations)

- *Let* $\mathsf{rm}_A : S \to S'$ *be a total function taking a (well-formed) declaration* $D \in S$ *and returning a (well formed) declaration* $D' \in S'$ *where* D' *is* D *with* A *removed, if* $A \in D$*; otherwise* $D' = D$.
- *Let* $\mathsf{perm}_\pi : S \to S'$ *be a total function that permutes the elements of a (well-formed) declaration* $D \in S$ *according to* π *to obtain a (well formed) declaration* $D' \in S'$.

Using these operations on declarations we state structural properties of declarations, later to be extended to contexts. These make no assumptions and give no guarantees about the schema of the context Γ, D and the resulting context

$\Gamma, f(D)$ where $f \in \{\mathsf{rm}_A, \mathsf{perm}_\pi\}$. In fact, we often want to use these properties when Γ satisfies some schema S, but D does not *yet* fit S; in this case, we apply an operation to D so that $\Gamma, f(D)$ *does* satisfy the schema S.

Since our context schema may contain alternatives, the function rm is defined via case-analysis covering all the possibilities, where we describe dropping all assumptions of a case using a dot, e.g., $\mathsf{is\_tm}\ x \mapsto \cdot$. For example:

- $\mathsf{rm}_{x:A} : S_{\alpha t} \to S_{\alpha x} = \lambda d.\mathsf{case}\ d\ \mathsf{of}\ \mathsf{is\_tp}\ \alpha \mapsto \mathsf{is\_tp}\ \alpha \mid \mathsf{is\_tm}\ y; y{:}A \mapsto \mathsf{is\_tm}\ y$
- $\mathsf{rm}_{\mathsf{is\_tm}\ x} : S_{\alpha x} \to S_{\alpha} = \lambda d.\mathsf{case}\ d\ \mathsf{of}\ \mathsf{is\_tp}\ \alpha \mapsto \mathsf{is\_tp}\ \alpha \mid \mathsf{is\_tm}\ y \mapsto \cdot$

Property 2 (Structural Properties of Declarations)

1. *Declaration Weakening:*

$$\frac{\Gamma, \mathsf{rm}_A(D), \Gamma' \vdash J}{\Gamma, D, \Gamma' \vdash J}\ \textit{d-wk}$$

2. *Declaration Strengthening:*

$$\frac{\Gamma, D, \Gamma' \vdash J}{\Gamma, \mathsf{rm}_A(D), \Gamma' \vdash J}\ \textit{d-str}\dagger$$

with the proviso (†) *that A is irrelevant to J and Γ'.*[3]

3. *Declaration Exchange:*

$$\frac{\Gamma, D, \Gamma' \vdash J}{\Gamma, \mathsf{perm}_\pi(D), \Gamma' \vdash J}\ \textit{d-exc}$$

The special case $\mathsf{rm}_A(A)$ drops A completely, since

$$\mathsf{rm}_A = \lambda d.\mathsf{case}\ d\ \mathsf{of}\ A \mapsto \cdot \mid \ldots$$

We treat $\Gamma, \cdot, \Gamma'$ as equivalent to Γ, Γ'. Hence, in the special case where we have $\Gamma, \mathsf{rm}_A(A), \Gamma'$, we obtain the well-known weakening and strengthening laws on contexts that are often stated as:

$$\frac{\Gamma, A, \Gamma' \vdash J}{\Gamma, \Gamma' \vdash J}\ \textit{str}\dagger \qquad \frac{\Gamma, \Gamma' \vdash J}{\Gamma, A, \Gamma' \vdash J}\ \textit{wk}$$

In contrast to the above, the general exchange property on blocks of declarations cannot be obtained “for free” from the above operations and we define it explicitly:

Property 3 (Exchange)

$$\frac{\Gamma, D', D, \Gamma' \vdash J}{\Gamma, D, D', \Gamma' \vdash J}\ \textit{exc}$$

with the proviso that the sub-context D, D' is well-formed.

[3] In practice, this may be done by maintaining a dependency call graph of all judgments.

Further, we state structural properties of *contexts* generically. To "strengthen" *all* declarations in a given context Γ, we simply write $\mathsf{rm}^*_A(\Gamma)$ using the $*$ superscript. More generally, by f^* with $f \in \{\mathsf{rm}_A,\ \mathsf{perm}_\pi\}$, we mean the *iteration* of the operation f over a context.

Property 4 (Structural Properties of Contexts)

1. *Context weakening*
$$\frac{\mathsf{rm}^*_A(\Gamma) \vdash J}{\Gamma \vdash J}\ \textit{c-wk}$$
2. *Context strengthening*
$$\frac{\Gamma \vdash J}{\mathsf{rm}^*_A(\Gamma) \vdash J}\ \textit{c-str}\dagger$$
with the proviso (†) *that declarations that are instances of A are irrelevant to J.*
3. *Context exchange*
$$\frac{\Gamma \vdash J}{\mathsf{perm}^*_\pi(\Gamma) \vdash J}\ \textit{c-exc}$$

Finally, by rm_D (resp. rm^*_D), we mean the iteration of rm_A (resp. rm^*_A) for every $A \in D$, while keeping the resulting declaration and the overall context well-formed, e.g. $\mathsf{rm}_{\mathsf{is\_tm}\ y;\, y:A}(\_) = \mathsf{rm}_{\mathsf{is\_tm}\ y}(\mathsf{rm}_{y:A}(\_))$. All the above properties are *admissible* with respect to those extended rm functions.

The following examples illustrate some of the subtleties of this machinery:

- $\Gamma, \mathsf{rm}_{x:A}(\mathsf{is\_tm}\ y; y{:}A) = \Gamma, \mathsf{is\_tm}\ y$. Bound variables in the annotation of rm can always be renamed so that they are consistent with the eigenvariables used in the declaration.
- $\mathsf{rm}^*_{\mathsf{is\_tm}\ x}(\mathsf{is\_tm}\ x_1, \mathsf{is\_tp}\ \alpha, \mathsf{is\_tp}\ \beta, \mathsf{is\_tm}\ x_2) = \mathsf{is\_tp}\ \alpha, \mathsf{is\_tp}\ \beta$. Here, the rm operation drops one of the alternatives in the schema $S_{\alpha x}$.
- $\mathsf{rm}^*_{y:A}(\mathsf{is\_tm}\ x_1; x_1{:}\mathsf{nat},\ \mathsf{is\_tm}\ x_2; x_2{:}\mathsf{bool},\ \mathsf{is\_tp}\ \alpha) = (\mathsf{is\_tm}\ x_1, \mathsf{is\_tm}\ x_2, \mathsf{is\_tp}\ \alpha)$. The schematic variable A occurring in the annotation of rm will be instantiated with nat when strengthening the block $\mathsf{is\_tm}\ x_1; x_1{:}\mathsf{nat}$ and similarly with bool.
- $\mathsf{rm}^*_{\mathsf{is\_tm}\ y;\, y:A}(\mathsf{is\_tp}\ \alpha,\ \mathsf{is\_tp}\ \beta) = (\mathsf{is\_tp}\ \alpha,\ \mathsf{is\_tp}\ \beta)$. A rm operation may leave a context unchanged.

We state next the substitution properties for assumptions. The *parametric substitution* property allows us to instantiate parameters, i.e., eigenvariables, in the context. For example, given $\mathsf{is\_tp}\ \alpha, \mathsf{is\_tp}\ \beta \vdash J$ and a type bool, we can obtain $\mathsf{is\_tp}\ \mathsf{bool}, \mathsf{is\_tp}\ \beta \vdash [\mathsf{bool}/\alpha]J$ by replacing α with bool. The *hypothetical substitution* property allows us to eliminate an atomic formula A that is part of a declaration D. For example, given $\mathsf{is\_tp}\ \mathsf{bool},\ \mathsf{is\_tp}\ \beta \vdash J$ and evidence that $\mathsf{is\_tp}\ \mathsf{bool}$, we can obtain $\mathsf{is\_tp}\ \beta \vdash J$. In type theory the two substitution properties collapse into one.

Property 5 (Substitution Properties)

- ***Hypothetical Substitution**:*
If $\Gamma_1, (D_1; A; D_2), \Gamma_2 \vdash J$ and $\Gamma_1, D_1 \vdash A$, then $\Gamma_1, (D_1; D_2), \Gamma_2 \vdash J$ provided that $D_1; D_2$ is a well-formed declaration in Γ_1.

– ***Parametric Substitution**:*
If $\Gamma_1, (D_1; is\_wf\ \ x; D_2), \Gamma_2 \vdash J$, then $\Gamma_1, (D_1; [t/x]D_2),\ [t/x]\Gamma_2 \vdash [t/x]J$ for any term t for which $\Gamma_1, D_1 \vdash is\_wf\ t$ holds.

While parametric and hypothetical substitution do not preserve schema satisfaction by definition, we typically use them in such a way that contexts continue to satisfy a given schema.

We close this section recalling that, although we concentrate in our benchmarks on describing object languages that satisfy structural properties usually associated with intuitionistic logic, we note that our terminology can be used to also characterize sub-structural object languages. In the case of a linear object language, we might choose to only use operations such as perm and omit operations such as rm so as to faithfully and adequately characterize the allowed context operations.

2.4 The Polymorphic Lambda-Calculus Revisited

In systems supporting HOAS, inference rules are usually expressed using an implicit-context representation as illustrated in Fig. 1. The need for explicit structured contexts, as discussed in Sects. 2.2 and 2.3, arises when performing meta-reasoning about the judgments expressed by these inference rules. In order to make the link, we revisit the example from Sect. 2.1 giving a presentation with explicit contexts, and then we make some preliminary remarks about context schemas and meta-reasoning. We will adopt the explicit-context representation of inference rules in the rest of the paper with the informal understanding of how to move between the implicit and explicit formulations.

Well-formed Types

$$\frac{\mathsf{is\_tp}\ \alpha \in \Gamma}{\Gamma \vdash \mathsf{is\_tp}\ \alpha}\ tp_v \qquad \frac{\Gamma \vdash \mathsf{is\_tp}\ A \quad \Gamma \vdash \mathsf{is\_tp}\ B}{\Gamma \vdash \mathsf{is\_tp}\ (\mathsf{arr} A\ B)}\ tp_{ar} \qquad \frac{\Gamma, \mathsf{is\_tp}\ \alpha \vdash \mathsf{is\_tp}\ A}{\Gamma \vdash \mathsf{is\_tp}\ (\mathsf{all}\ \alpha.\ A)}\ tp_{al}$$

Well-formed Terms

$$\frac{\mathsf{is\_tm}\ x \in \Gamma}{\Gamma \vdash \mathsf{is\_tm}\ x}\ tm_v \qquad \frac{\Gamma, \mathsf{is\_tm}\ x \vdash \mathsf{is\_tm}\ M}{\Gamma \vdash \mathsf{is\_tm}\ (\mathsf{lam}\ x.\ M)}\ tm_l \qquad \frac{\Gamma, \mathsf{is\_tp}\ \alpha \vdash \mathsf{is\_tm}\ M}{\Gamma \vdash \mathsf{is\_tm}\ (\mathsf{tlam}\ \alpha.\ M)}\ tm_{tl}$$

$$\frac{\Gamma \vdash \mathsf{is\_tm}\ M_1 \qquad \Gamma \vdash \mathsf{is\_tm}\ M_2}{\Gamma \vdash \mathsf{is\_tm}\ (\mathsf{app}\ M_1\ M_2)}\ tm_a \qquad \frac{\Gamma \vdash \mathsf{is\_tm}\ M \qquad \Gamma \vdash \mathsf{is\_tp}\ A}{\Gamma \vdash \mathsf{is\_tm}\ (\mathsf{tapp}\ M\ A)}\ tm_{ta}$$

Typing for the Polymorphic λ-Calculus

$$\frac{x{:}B \in \Gamma}{\Gamma \vdash x : B}\ of_v \qquad \frac{\Gamma, \mathsf{is\_tp}\ \alpha \vdash M : B}{\Gamma \vdash \mathsf{tlam}\ \alpha.\ M : \mathsf{all}\ \alpha.\ B}\ of_{tl} \qquad \frac{\Gamma \vdash M : \mathsf{all}\ \alpha.\ B \quad \Gamma \vdash \mathsf{is\_tp}\ B}{\Gamma \vdash (\mathsf{tapp}\ M\ B) : [B/\alpha]A}\ of_{ta}$$

$$\frac{\Gamma, \mathsf{is\_tm}\ x; x{:}A \vdash M : B}{\Gamma \vdash \mathsf{lam}\ x.\ M : \mathsf{arr} A\ B}\ of_l \qquad \frac{\Gamma \vdash M : \mathsf{arr} B\ A \qquad \Gamma \vdash N : B}{\Gamma \vdash (\mathsf{app}\ M\ N) : A}\ of_a$$

In this formulation, and differently from the implicit one, we have a base case for variables. Here, to look up an assumption in a context, we simply write $A \in \Gamma$, meaning that there is some block D in context Γ such that $A \in D$. For example $x{:}B \in \Gamma$ holds if Γ contains block $\mathsf{is\_tm}\ x; x{:}B$. We will also overload the notation and write $D \in \Gamma$ to indicate that Γ contains the entire block D. We recall the distinction between the comma used to separate blocks, and the semi-colon used to separate atoms within blocks, as seen in the of_l rule, for example. The assumption that all variables occurring in contexts are distinct from one another is silently preserved by the implicit proviso in rules that extend the context, where we rename the bound variable if already present.

Note that we use a generic Γ for the context appearing in these rules, whereas the reader may have expected this to be, for example, $\Phi_{\alpha t}$ having schema $S_{\alpha t}$ in the typing rules. In fact, we take a more liberal approach, where we pass to the rules *any* context that can be seen as a *weakening* of $\Phi_{\alpha t}$; in other words, any Γ such that there exists a D for which $\mathsf{rm}^*_D(\Gamma) = \Phi_{\alpha t}$.

Suppose now, to fix ideas, that $\Phi_{\alpha t} \vdash M : B$ holds. By convention, we implicitly assume that both B and M are well-formed, which means that $\Phi_{\alpha t} \vdash \mathsf{is\_tp}\ B$ and $\Phi_{\alpha t} \vdash \mathsf{is\_tm}\ M$. In fact, we can define functions $\mathsf{rm}^*_{x:C}$ and $\mathsf{rm}^*_{\mathsf{is\_tm}\ x; x:C}$, use them to define strengthened contexts $\Phi_{\alpha x}$ and Φ_α, and apply the *c-str* rule to conclude the following:

1. $\Phi_{\alpha x} := \mathsf{rm}^*_{x:C}(\Phi_{\alpha t})$, $\quad \Phi_{\alpha x}$ $\mathsf{has\_schema}$ $S_{\alpha x}$, $\quad$ and $\Phi_{\alpha x} \vdash \mathsf{is\_tm}\ M$;
2. $\Phi_{\alpha} := \mathsf{rm}^*_{\mathsf{is\_tm}\ x; x:C}(\Phi_{\alpha t})$, $\quad \Phi_{\alpha}$ $\mathsf{has\_schema}$ S_{α}, $\quad$ and $\Phi_{\alpha} \vdash \mathsf{is\_tp}\ B$.

2.5 Generalized Contexts vs. Context Relations

As an alternative to using functions such as $\mathsf{rm}^*_{x:C}$ in item (1), we may adopt the more suggestive notation $\Phi_{\alpha x} \sim \Phi_{\alpha t}$, using inference rules for the context relation corresponding to the graph of the function $\lambda d.\mathsf{case}\ d\ \mathsf{of}\ \mathsf{is\_tp}\ \alpha \mapsto \mathsf{is\_tp}\ \alpha \mid \mathsf{is\_tm}\ x; x{:}C \mapsto \mathsf{is\_tm}\ x$:

$$\frac{}{\cdot \sim \cdot} \qquad \frac{\Phi_{\alpha x} \sim \Phi_{\alpha t}}{(\Phi_{\alpha x}, \mathsf{is\_tp}\ \alpha) \sim (\Phi_{\alpha t}, \mathsf{is\_tp}\ \alpha)} \qquad \frac{\Phi_{\alpha x} \sim \Phi_{\alpha t}}{(\Phi_{\alpha x}, \mathsf{is\_tm}\ x) \sim (\Phi_{\alpha t}, \mathsf{is\_tm}\ x; x{:}B)}$$

Similarly, an alternative to $\mathsf{rm}^*_{\mathsf{is\_tm}\ x; x:C}$ in item (2) is the following context relation:

$$\frac{}{\cdot \sim \cdot} \qquad \frac{\Phi_{\alpha} \sim \Phi_{\alpha t}}{(\Phi_{\alpha}, \mathsf{is\_tp}\ \alpha) \sim (\Phi_{\alpha t}, \mathsf{is\_tp}\ \alpha)} \qquad \frac{\Phi_{\alpha} \sim \Phi_{\alpha t}}{\Phi_{\alpha} \sim (\Phi_{\alpha t}, \mathsf{is\_tm}\ x; x{:}B)}$$

The above two statements can now be restated using these relations. Given $\Phi_{\alpha t}$, let $\Phi_{\alpha x}$ and Φ_α be the unique contexts such that:

1. $\Phi_{\alpha x} \sim \Phi_{\alpha t}$, $\quad \Phi_{\alpha x}$ $\mathsf{has\_schema}$ $S_{\alpha x}$, $\quad$ and $\Phi_{\alpha x} \vdash \mathsf{is\_tm}\ M$;
2. $\Phi_{\alpha} \sim \Phi_{\alpha t}$, $\quad \Phi_{\alpha}$ $\mathsf{has\_schema}$ S_{α}, $\quad$ and $\Phi_{\alpha} \vdash \mathsf{is\_tp}\ B$.

When stating and proving properties, we often relate two judgments to each other, where each one has its own contexts. For example, we may want to prove statements such as "if $\Phi_{\alpha x} \vdash J_1$ then $\Phi_{\alpha t} \vdash J_2$." The question is how we achieve that. In the benchmarks in this paper, we consider two approaches:

1. We reinterpret the statement in the *smallest* context that collects all relevant assumptions; we call this the *generalized context* approach (G). In this case, we reinterpret the above statement about J_1 in a context containing additional assumptions about typing, which in this case is $\Phi_{\alpha t}$, yielding:

"if $\Phi_{\alpha t} \vdash J_1$ then $\Phi_{\alpha t} \vdash J_2$."

2. We state how two (or more) contexts are *related*; we call this the *context relations* approach (R). Here, we define context relations such as those above and use them *explicitly* in the statements of theorems. In this case, we use $\Phi_{\alpha x} \sim \Phi_{\alpha t}$ yielding:

   "if $\Phi_{\alpha x} \vdash J_1$ and $\Phi_{\alpha x} \sim \Phi_{\alpha t}$ then $\Phi_{\alpha t} \vdash J_2$."

   Note that here too we "minimize" the relations, in the sense of relating the smallest possible contexts where the relevant judgments make sense.

2.6 Context Promotion and Linear Extension of Contexts and Schemas

Another common idiom in meta-reasoning occurs when we have established a property for a particular context and we would like to use this property subsequently in a more general context. Assume that we have proven a lemma about types in context Φ_α of the form "if $\Phi_\alpha \vdash J_1$ then $\Phi_\alpha \vdash J_2$." We now want to *use* this lemma in a proof about terms, that is where we have a context $\Phi_{\alpha x}$ and $\Phi_{\alpha x} \vdash J_1$. We may need to *promote* this lemma, and prove: "if $\Phi_{\alpha x} \vdash J_1$ then $\Phi_{\alpha x} \vdash J_2$." We will see several examples of such promotion lemmas in Sect. 3.

Finally, to structure our subsequent discussion, it is useful to introduce some additional terminology regarding context relationships, where we use "relationship" in contrast to the more specific notion of "context relation."

- *Linear extension of a declaration*: a declaration D_2 is a *linear extension* of a declaration D_1, if every atom in the declaration D_1 is a member of the declaration D_2.
- *Linear extension of a schema*: a schema S_2 is a *linear extension* of a schema S_1, if every declaration in S_1 is a linear extension of a declaration in S_2. For example $S_{\alpha t}$ is a linear extension of $S_{\alpha x}$.

Given a context Φ_1 of schema S_1 and a context Φ_2 of schema S_2 where S_2 is a linear extension of S_1, we say that Φ_2 is a linear extension of Φ_1 (i.e., linear context extension). Of course, sometimes declarations, schemas and contexts are not related linearly. For example, we may have a schema S_2 and a schema S_3 both of which are linear extensions of S_1; however, S_2 is not a linear extension of S_3 (or vice versa). In this case, we say S_2 and S_3 are *non-linear extensions* of each other and they share a most specific common fragment.

3 Benchmarks

In this section, we present several case studies establishing proofs of various properties of the lambda-calculus. We have structured this section around the different shapes and properties of contexts, namely:

1. Basic linear context extensions: We consider here contexts containing no alternatives. We refer to such contexts as *basic*. We discuss context membership and revisit structural properties such as weakening and strengthening.
2. Linear context extensions with alternative declarations.

3. Non-linear context extensions: We consider more complex relationships between contexts and discuss how our proofs involving weakening and strengthening change.
4. Order: We consider how the ordered structure of contexts impacts proofs relying on exchange.
5. Uniqueness: We consider here a case study which highlights how the issue of distinctness of all variable declarations in a context arises in proofs.
6. Substitution: Finally, we exhibit the fundamental properties of hypothetical and parametric substitution.

The benchmark problems are purposefully *simple*; they are designed to be easily understood so that one can quickly appreciate the capabilities and trade-offs of the different systems in which they can be implemented. Yet we believe they are representative of the issues and problems arising when encoding formal systems and reasoning about them. We will subsequently discuss both the G approach and the R approach and comment on the trade-offs and differences in proofs depending on the chosen approach.

3.1 Basic Linear Context Extension

We concentrate in this section on contexts with simple schemas consisting of a single declaration. We aim to show the basic building blocks of reasoning over open terms: namely what a context looks like and the structure of an inductive proof. For the latter, we focus on the case analysis and, at the risk of being pedantic, the precise way in which the induction hypothesis is applied.

We start with a very simple judgment: *algorithmic equality* for the untyped lambda-calculus, written ($\mathsf{aeq}\ M\ N$), also known as *copy* clauses (see Miller, 1991). We say that two terms are algorithmically equal provided they have the same structure with respect to the constructors.

Algorithmic Equality

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

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

The context schemas needed for reasoning about this judgment are the following:

$$\text{Context Schemas } \begin{aligned} S_x &:= \mathsf{is\_tm}\ x \\ S_{xa} &:= \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \end{aligned}$$

where a context Φ_{xa} satisfying S_{xa} is the smallest possible context in which such an equality judgment can hold. Thus, as discussed in the previous section, when writing judgment $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$, we assume that $\Phi_{xa} \vdash \mathsf{is\_tm}\ M$ and $\Phi_{xa} \vdash \mathsf{is\_tm}\ N$ hold, and thus also $\Phi_x \vdash \mathsf{is\_tm}\ M$ and $\Phi_x \vdash \mathsf{is\_tm}\ N$ hold by employing an implicit *c-str* (using $\mathsf{rm}^*_{\mathsf{aeq}\ x\ x}$). We note that both contexts Φ_x and Φ_{xa} are simple contexts consisting of one declaration block. Moreover, S_x is a sub-schema of S_{xa} and therefore the context Φ_{xa} is a linear extension of the context Φ_x.

In view of the pedagogical nature of this subsection and also of the content of Sect. 3.3, which will build on this example, we start with a straightforward property: algorithmic equality is reflexive. This property should follow by induction on M (via the well-formed term judgment, which is not shown, but uses the obvious subset of the rules in Sect. 2.4). However, the question of which contexts the two judgments should be stated in arises immediately; recall that we want to prove "if $\Gamma_1 \vdash \mathsf{is\_tm}\ M$ then $\Gamma_2 \vdash \mathsf{aeq}\ M\ M$." Γ_2 should be a context satisfying S_{xa} since the definition of this schema came directly from the inference rules of this judgment. The form that Γ_1 should take is less clear. The main requirement comes from the base case, where we must know that for every assumption $\mathsf{is\_tm}\ x$ in Γ_1 there exists a corresponding assumption $\mathsf{aeq}\ x\ x$ in Γ_2. The answer differs depending on whether we choose the R approach or the G approach. We discuss each in turn below.

3.1.1 Context Relations, R Version

The relation needed here is $\Phi_x \sim \Phi_{xa}$, defined as follows:

Context Relation

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

Note 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}. This is a property which needs to be established separately, so at the risk of redundancy, we state it as a "member" lemma.

Lemma 6 (Context Membership) *$\Phi_x \sim \Phi_{xa}$ implies that $\mathsf{is\_tm}\ x \in \Phi_x$ iff $\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \in \Phi_{xa}$.*

Proof By induction on $\Phi_x \sim \Phi_{xa}$.

Theorem 7 (Admissibility of Reflexivity, R Version) *Assume $\Phi_x \sim \Phi_{xa}$. If $\Phi_x \vdash \mathsf{is\_tm}\ M$ then $\Phi_{xa} \vdash \mathsf{aeq}\ M\ M$.*

Proof By induction on the derivation $\mathcal{D} :: \Phi_x \vdash \mathsf{is\_tm}\ M$.
Case:

$$\mathcal{D} = \frac{\mathsf{is\_tm}\ x \in \Phi_x}{\Phi_x \vdash \mathsf{is\_tm}\ x}\ tm_v$$

$\mathsf{is\_tm}\ x \in \Phi_x$	by rule premise
$\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \in \Phi_{xa}$	by Lemma 6
$\Phi_{xa} \vdash \mathsf{aeq}\ x\ x$	by rule ae_v

Case:

$$\mathcal{D} = \frac{\begin{matrix}\mathcal{D}_1 \\ \Phi_x \vdash \mathsf{is\_tm}\ M_1\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Phi_x \vdash \mathsf{is\_tm}\ M_2\end{matrix}}{\Phi_x \vdash \mathsf{is\_tm}\ (\mathsf{app}\ M_1\ M_2)}\ tm_a$$

$\Phi_x \vdash \mathsf{is\_tm}\ M_1$	sub-derivation $\mathcal{D}_1$
$\Phi_{xa} \vdash \mathsf{aeq}\ M_1\ M_1$	by IH
$\Phi_x \vdash \mathsf{is\_tm}\ M_2$	sub-derivation $\mathcal{D}_2$

$\Phi_{xa} \vdash \mathsf{aeq}\ M_2\ M_2$ by IH
$\Phi_{xa} \vdash \mathsf{aeq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ M_1\ M_2)$ by rule ae_a

Case:

$$\mathcal{D} = \cfrac{\begin{array}{c}\mathcal{D}'\\ \Phi_x, \mathsf{is\_tm}\ x \vdash \mathsf{is\_tm}\ M\end{array}}{\Phi_x \vdash \mathsf{is\_tm}\ (\mathsf{lam}\, x.\, M)}\ tm_l$$

$\Phi_x, \mathsf{is\_tm}\ x \vdash \mathsf{is\_tm}\ M$ sub-derivation $\mathcal{D}'$
$\Phi_x \sim \Phi_{xa}$ by assumption
$(\Phi_x, \mathsf{is\_tm}\ x) \sim (\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x)$ by rule $crel_{xa}$
$\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M\ M$ by IH
$\Phi_{xa} \vdash \mathsf{aeq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, M)$ by rule ae_l.

3.1.2 Generalized Contexts, G Version

In this example, since S_{xa} includes all assumptions in S_x, S_{xa} will serve as the schema of our generalized context.

Theorem 8 (Admissibility of Reflexivity, G Version) *If* $\Phi_{xa} \vdash \mathsf{is\_tm}\ M$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ M$.

Proof By induction on the derivation $\mathcal{D} :: \Phi_{xa} \vdash \mathsf{is\_tm}\ M$.
Case:

$$\mathcal{D} = \cfrac{\mathsf{is\_tm}\ x \in \Phi_{xa}}{\Phi_{xa} \vdash \mathsf{is\_tm}\ x}\ tm_v$$

$\mathsf{is\_tm}\ x \in \Phi_{xa}$ by rule premise
Φ_{xa} contains block $(\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x)$ by definition of S_{xa}
$\Phi_{xa} \vdash \mathsf{aeq}\ x\ x$ by rule ae_v

Case:

$$\mathcal{D} = \cfrac{\begin{array}{c}\mathcal{D}_1\\ \Phi_{xa} \vdash \mathsf{is\_tm}\ M_1\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Phi_{xa} \vdash \mathsf{is\_tm}\ M_2\end{array}}{\Phi_{xa} \vdash \mathsf{is\_tm}\ (\mathsf{app}\ M_1\ M_2)}\ tm_a$$

$\Phi_{xa} \vdash \mathsf{aeq}\ M_1\ M_1$ by IH on $\mathcal{D}_1$
$\Phi_{xa} \vdash \mathsf{aeq}\ M_2\ M_2$ by IH on $\mathcal{D}_2$
$\Phi_{xa} \vdash \mathsf{aeq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ M_1\ M_2)$ by rule ae_a

Case:

$$\mathcal{D} = \cfrac{\begin{array}{c}\mathcal{D}'\\ \Phi_{xa}, \mathsf{is\_tm}\ x \vdash \mathsf{is\_tm}\ M\end{array}}{\Phi_{xa} \vdash \mathsf{is\_tm}\ (\mathsf{lam}\, x.\, M)}\ tm_l$$

$\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{is\_tm}\ M$ by *d-wk* on $\mathcal{D}'$
$\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M\ M$ by IH
$\Phi_{xa} \vdash \mathsf{aeq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, M)$ by rule ae_l

Note that the application cases of Theorems 7 and 8 are the same except for the context used for the well-formed term judgment. The lambda case here, on the other hand, requires an additional weakening step. In particular, *d-wk* is used to add an atom to form the declaration needed for schema S_{xa}. The context before applying weakening does not satisfy this schema, and the induction hypothesis cannot be applied until it does.

We end this subsection, stating the remaining properties needed to establish that algorithmic equality is indeed a congruence, which we will prove in Sect. 3.3. Since the proof involves only Φ_{xa}, the two approaches (R & G) collapse.

Lemma 9 (Context Inversion) *If* $\mathsf{aeq}\ M\ N \in \Phi_{xa}$ *then* $M = N$.

Proof Induction on $\mathsf{aeq}\ M\ N \in \Phi_{xa}$.

Theorem 10 (Admissibility of Symmetry and Transitivity)

1. *If* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ N\ M$.
2. *If* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ L$ *and* $\Phi_{xa} \vdash \mathsf{aeq}\ L\ N$ *then* $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$.

Proof Induction on the given derivation using Lemma 9 in the variable case.

3.2 Linear Context Extensions with Alternative Declarations

We extend our algorithmic equality case study to the polymorphic lambda-calculus, highlighting the situation where judgments induce context schemas with *alternatives*. We accordingly add the judgment for *type equality*, $\mathsf{atp}\ A\ B$, noting that the latter can be defined independently of term equality. In other words $\mathsf{aeq}\ M\ N$ depends on $\mathsf{atp}\ A\ B$, but not vice-versa. In addition to S_α and $S_{\alpha x}$ introduced in Sect. 2, the following new context schemas are also used here:

$$\begin{aligned} S_{atp} &:= \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha \\ S_{aeq} &:= \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha\ +\ \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \end{aligned}$$

The rules for the two equality judgments extend those given in Sect. 3.1. The additional rules are stated below.

Algorithmic Equality for the Polymorphic Lambda-Calculus

$$\ldots$$

$$\frac{\Gamma, \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha \vdash \mathsf{aeq}\ M\ N}{\Gamma \vdash \mathsf{aeq}\ (\mathsf{tlam}\,\alpha.\,M)\ (\mathsf{tlam}\,\alpha.\,N)}\ ae_{tl}$$

$$\frac{\Gamma \vdash \mathsf{aeq}\ M\ N \quad \Gamma \vdash \mathsf{atp}\ A\ B}{\Gamma \vdash \mathsf{aeq}\ (\mathsf{tapp}\ M\ A)\ (\mathsf{tapp}\ N\ B)}\ ae_{ta}$$

$$\frac{\mathsf{atp}\ \alpha\ \alpha \in \Gamma}{\Gamma \vdash \mathsf{atp}\ \alpha\ \alpha}\ at_\alpha$$

$$\frac{\Gamma, \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha \vdash \mathsf{atp}\ A\ B}{\Gamma \vdash \mathsf{atp}\ (\mathsf{all}\,\alpha.\,A)\ (\mathsf{all}\,\alpha.\,B)}\ at_{al} \qquad \frac{\Gamma \vdash \mathsf{atp}\ A_1\ B_1 \quad \Gamma \vdash \mathsf{atp}\ A_2\ B_2}{\Gamma \vdash \mathsf{atp}\ (\mathsf{arr} A_1\ A_2)\ (\mathsf{arr} B_1\ B_2)}\ at_a$$

We show again the admissibility of reflexivity. We start with the G version this time.

3.2.1 G Version

We first state and prove the admissibility of reflexivity for types, which we then use in the proof of admissibility of reflexivity for terms. The schema for the generalized context for the former is S_{atp} since the statement and proof do not depend on terms. The schema for the latter is S_{aeq}.

Theorem 11 (Admissibility of Reflexivity for Types, G Version)
If $\Phi_{atp} \vdash \mathsf{is\_tp}\ A$ *then* $\Phi_{atp} \vdash \mathsf{atp}\ A\ A$.

The proof is exactly the same as the proof of Theorem 8, modulo replacing app and lam with arr and all, respectively, and using the corresponding rules.

As we have already mentioned in Sect. 2, it is often the case that we need to appeal to a lemma in a context that is different from the context where it was proved. A concrete example is the above lemma, which is stated in context Φ_{atp}, but is needed in the proof of the next theorem in the larger context Φ_{aeq}. To illustrate, we state and prove the necessary *promotion* lemma here.

Lemma 12 (G-Promotion for Type Reflexivity)
If $\Phi_{aeq} \vdash \mathsf{is\_tp}\ A$ *then* $\Phi_{aeq} \vdash \mathsf{atp}\ A\ A$.

Proof

$\Phi_{aeq} \vdash \mathsf{is\_tp}\ A$	by assumption
$\Phi_{atp} \vdash \mathsf{is\_tp}\ A$	by *c-str*
$\Phi_{atp} \vdash \mathsf{atp}\ A\ A$	by Theorem 11
$\Phi_{aeq} \vdash \mathsf{atp}\ A\ A$	by *c-wk*

In general, proofs of promotion lemmas require applications of *c-str* and *c-wk* which perform a uniform modification to an entire context. In contrast, the abstraction cases in proofs such as the lambda case of Theorem 8 require *d-wk* to add atoms to a single declaration. The particular function used here is $\mathsf{rm}^*_{\mathsf{is\_tm}\ x;\mathsf{aeq}\ x\ x}$, which drops an entire alternative from Φ_{aeq} to obtain Φ_{atp} and leaves the other alternative unchanged. The combination of *c-str* and *c-wk* in proofs of promotion lemmas is related to *subsumption* (see Harper and Licata, 2007).

Note that we could omit Theorem 11 and instead prove Lemma 12 directly, removing the need for a promotion lemma. For modularity purposes, we adopt the approach that we state each theorem in the smallest possible context in which it is valid. This particular lemma, for example, will be needed in an even bigger context than Φ_{aeq} in Sect. 3.3. In general, we do not want the choice of context in the statement of a lemma to depend on later theorems whose proofs require this lemma. Instead, we choose the smallest context and state and prove promotion lemmas where needed.

Theorem 13 (Admissibility of Reflexivity for Terms, G Version)
If $\Phi_{aeq} \vdash \mathsf{is\_tm}\ M$ *then* $\Phi_{aeq} \vdash \mathsf{aeq}\ M\ M$.

Proof Again, the proof is by induction on the given well-formed term derivation, in this case $\mathcal{D} :: \Phi_{aeq} \vdash \mathsf{is\_tm}\ M$, and is similar to the proof of Theorem 8. We show the case for application of terms to types.

Case:

$$\mathcal{D} = \frac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Phi_{aeq} \vdash \mathsf{is\_tm}\ M & \Phi_{aeq} \vdash \mathsf{is\_tp}\ A \end{array}}{\Phi_{aeq} \vdash \mathsf{is\_tm}\ (\mathsf{tapp}\ M\ A)}$$

$\Phi_{aeq} \vdash \mathsf{aeq}\ M\ M$ by IH on $\mathcal{D}_1$
$\Phi_{aeq} \vdash \mathsf{atp}\ A\ A$ by Lemma 12 on conclusion of $\mathcal{D}_2$
$\Phi_{aeq} \vdash \mathsf{aeq}\ (\mathsf{tapp}\ M\ A)\ (\mathsf{tapp}\ M\ A)$ by rule ae_{ta}

3.2.2 R Version

We introduce four context relations $\Phi_\alpha \sim \Phi_{atp}$, $\Phi_{\alpha x} \sim \Phi_{aeq}$, $\Phi_{\alpha x} \sim \Phi_\alpha$, and $\Phi_{aeq} \sim \Phi_{atp}$. We define the first two as follows (where we omit the inference rules for the base cases).

Context Relations

$$\frac{\Phi_\alpha \sim \Phi_{atp}}{\Phi_\alpha, \mathsf{is\_tp}\ \alpha \sim \Phi_{atp}, \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha}$$

$$\frac{\Phi_{\alpha x} \sim \Phi_{aeq}}{\Phi_{\alpha x}, \mathsf{is\_tm}\ x \sim \Phi_{aeq}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x} \qquad \frac{\Phi_{\alpha x} \sim \Phi_{aeq}}{\Phi_{\alpha x}, \mathsf{is\_tp}\ \alpha \sim \Phi_{aeq}, \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha}$$

Note that $\Phi_{\alpha x} \sim \Phi_{aeq}$ is the extension of $\Phi_x \sim \Phi_{xa}$ with one additional case for equality for types.[4] We also omit the (obvious) inference rules defining $\Phi_{\alpha x} \sim \Phi_\alpha$ and $\Phi_{aeq} \sim \Phi_{atp}$, and instead note that they correspond to the graphs of the following two functions, respectively, which simply remove one of the two schema alternatives:

$$\mathsf{rm}^*_{\mathsf{is\_tm}\ x} = \lambda d.\mathsf{case}\ d\ \mathsf{of}\ \mathsf{is\_tp}\ \alpha \mapsto \mathsf{is\_tp}\ \alpha \mid \mathsf{is\_tm}\ x \mapsto \cdot$$
$$\mathsf{rm}^*_{\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x} = \lambda d.\mathsf{case}\ d\ \mathsf{of}\ \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha \mapsto \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha \mid \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \mapsto \cdot$$

We start with the theorem for types again, whose proof is similar to the R version of the previous example (Theorem 7) and is therefore omitted.

Theorem 14 (Admissibility of Reflexivity for Types, R Version)
Let $\Phi_\alpha \sim \Phi_{atp}$. If $\Phi_\alpha \vdash \mathsf{is\_tp}\ A$ then $\Phi_{atp} \vdash \mathsf{atp}\ A\ A$.

Lemma 15 (Relational Strengthening) *Let $\Phi_{\alpha x} \sim \Phi_{aeq}$. Then there exist contexts Φ_α and Φ_{atp} such that $\Phi_{\alpha x} \sim \Phi_\alpha$, $\Phi_{aeq} \sim \Phi_{atp}$, and $\Phi_\alpha \sim \Phi_{atp}$.*

Proof By induction on the given derivation of $\Phi_{\alpha x} \sim \Phi_{aeq}$.

We again need a promotion lemma, this time involving the context relation.

Lemma 16 (R-Promotion for Type Reflexivity)
Let $\Phi_{\alpha x} \sim \Phi_{aeq}$. If $\Phi_{\alpha x} \vdash \mathsf{is\_tp}\ A$ then $\Phi_{aeq} \vdash \mathsf{atp}\ A\ A$.

[4] Again, we remark on our policy to use the smallest contexts possible for modularity reasons. Otherwise, we could have omitted the $\Phi_\alpha \sim \Phi_{atp}$ relation, and stated the next theorem using $\Phi_{\alpha x} \sim \Phi_{aeq}$.

Proof

$\Phi_{\alpha x} \vdash \textsf{is\_tp}\ A$	by assumption
$\Phi_{\alpha} \vdash \textsf{is\_tp}\ A$	by *c-str*
$\Phi_{\alpha x} \sim \Phi_{aeq}$	by assumption
$\Phi_{\alpha} \sim \Phi_{atp}$	by relational strengthening (Lemma 15)
$\Phi_{atp} \vdash \textsf{atp}\ A\ A$	by Theorem 14
$\Phi_{aeq} \vdash \textsf{atp}\ A\ A$	by *c-wk*

Theorem 17 (Admissibility of Reflexivity for Terms, R Version) *Let $\Phi_{\alpha x} \sim \Phi_{aeq}$. If $\Phi_{\alpha x} \vdash \textsf{is\_tm}\ M$ then $\Phi_{aeq} \vdash \textsf{aeq}\ M\ M$.*

Proof Again, the proof is by induction on the given derivation. Most cases are similar to the analogous cases in the proof of the R version for the monomorphic case (Theorem 7) and the G version for types in the polymorphic case (Theorem 11). We show again the case for application of terms to types to compare with the G version.

Case:

$$\mathcal{D} = \frac{\begin{matrix}\mathcal{D}_1 \\ \Phi_{\alpha x} \vdash \textsf{is\_tm}\ M\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Phi_{\alpha x} \vdash \textsf{is\_tp}\ A\end{matrix}}{\Phi_{\alpha x} \vdash \textsf{is\_tm}\ (\textsf{tapp}\ M\ A)}$$

$\Phi_{\alpha x} \sim \Phi_{aeq}$	by assumption
$\Phi_{\alpha x} \vdash \textsf{is\_tm}\ M$	sub-derivation $\mathcal{D}_1$
$\Phi_{aeq} \vdash \textsf{aeq}\ M\ M$	by IH
$\Phi_{\alpha x} \vdash \textsf{is\_tp}\ A$	sub-derivation $\mathcal{D}_2$
$\Phi_{aeq} \vdash \textsf{is\_tp}\ A$	by Lemma 16
$\Phi_{aeq} \vdash \textsf{aeq}\ (\textsf{tapp}\ M\ A)\ (\textsf{tapp}\ M\ A)$	by rule ae_{ta}

3.3 Non-Linear Context Extensions

We return to the untyped lambda-calculus of Sect. 3.1 and establish the equivalence between the algorithmic definition of equality defined previously, and declarative equality $\Phi_{xd} \vdash \textsf{deq}\ M\ N$, which includes reflexivity, symmetry and transitivity in addition to the congruence rules.[5]

[5] We acknowledge that this definition of declarative equality has a degree of redundancy: the assumption $\textsf{deq}\ x\ x$ in rule de_l is not needed, since rule de_r plays the variable role. However, it yields an interesting generalized context schema, which exhibits issues that would otherwise require more complex case studies.

Declarative Equality

$$\frac{\mathsf{deq}\ x\ x \in \Gamma}{\Gamma \vdash \mathsf{deq}\ x\ x}\ de_v \qquad \frac{\Gamma, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x \vdash \mathsf{deq}\ M\ N}{\Gamma \vdash \mathsf{deq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)}\ de_l$$

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

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

Context Schema $S_{xd} ::= \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x$

We now investigate the interesting part of the equivalence, namely that when we have a proof of ($\mathsf{deq}\ M\ N$) then we also have a proof of ($\mathsf{aeq}\ M\ N$). We show the G version first.

3.3.1 G Version

Here, a generalized context must combine the atoms of Φ_{xa} and Φ_{xd} into one declaration:

$$\text{Generalized Context Schema } S_{da} := \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x; \mathsf{aeq}\ x\ x$$

The following lemma promotes Theorems 8 and 10 to the "bigger" generalized context.

Lemma 18 (G-Promotion for Reflexivity, Symmetry, and Transitivity)

1. *If* $\Phi_{da} \vdash \mathsf{is\_tm}\ M$*, then* $\Phi_{da} \vdash \mathsf{aeq}\ M\ M$*.*
2. *If* $\Phi_{da} \vdash \mathsf{aeq}\ M\ N$*, then* $\Phi_{da} \vdash \mathsf{aeq}\ N\ M$*.*
3. *If* $\Phi_{da} \vdash \mathsf{aeq}\ M\ L$ *and* $\Phi_{da} \vdash \mathsf{aeq}\ L\ N$*, then* $\Phi_{da} \vdash \mathsf{aeq}\ M\ N$*.*

Proof Similar to the proof of Theorem 12 where the application of *c-str* transforms a context Φ_{da} to Φ_{xa} by considering each block of the form ($\mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x; \mathsf{aeq}\ x\ x$) and removing ($\mathsf{deq}\ x\ x$).

Theorem 19 (Completeness, G Version)
If $\Phi_{da} \vdash \mathsf{deq}\ M\ N$ *then* $\Phi_{da} \vdash \mathsf{aeq}\ M\ N$*.*

Proof By induction on the derivation $\mathcal{D} :: \Phi_{da} \vdash \mathsf{deq}\ M\ N$. We only show some cases.
Case:

$$\mathcal{D} = \frac{}{\Phi_{da} \vdash \mathsf{deq}\ M\ M}\ de_r$$

$\Phi_{da} \vdash \mathsf{is\_tm}\ M$	by (implicit) assumption
$\Phi_{da} \vdash \mathsf{aeq}\ M\ M$	by Lemma 18 (1)

Case:

$$\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Phi_{da} \vdash \mathsf{deq}\ M\ L & \Phi_{da} \vdash \mathsf{deq}\ L\ N\end{array}}{\Phi_{da} \vdash \mathsf{deq}\ M\ N}\ de_t$$

$\Phi_{da} \vdash \mathsf{aeq}\ M\ L$ and $\Phi_{da} \vdash \mathsf{aeq}\ L\ N$ by IH on $\mathcal{D}_1$ and $\mathcal{D}_2$
$\Phi_{da} \vdash \mathsf{aeq}\ M\ N$ by Lemma 18 (3)

Case:

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}' \\ \Phi_{da}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x \vdash \mathsf{deq}\ M\ N\end{array}}{\Phi_{da} \vdash \mathsf{deq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)}\ de_l$$

$\Phi_{da}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{deq}\ M\ N$ by *d-wk* on $\mathcal{D}'$
$\Phi_{da}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M\ N$ by IH
$\Phi_{da}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M\ N$ by *d-str*
$\Phi_{da} \vdash \mathsf{aeq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)$ by rule ae_l

The symmetry case is not shown, but also requires promotion, via Lemma 18 (2). Note that the de_l case requires both *d-str* and *d-wk*. In contrast, the binder cases for the G versions of the previous examples (Theorems 8, 11, and 13) required only *d-wk*. The need for both arises from the fact that the generalized context is a non-linear extension of two contexts, i.e., it is not the same as either one of the two contexts it combines.

3.3.2 R Version

The context relation required here is $\Phi_{xa} \sim \Phi_{xd}$:

Context Relation

$$\dfrac{\Phi_{xa} \sim \Phi_{xd}}{\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \sim \Phi_{xd}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x}\ crel_{ad}$$

As in Sect. 3.2, we need the appropriate promotion lemma, which again requires a relation strengthening lemma:

Lemma 20 (Relational Strengthening) *Let $\Phi_{xa} \sim \Phi_{xd}$. Then there exists a context Φ_x such that $\Phi_x \sim \Phi_{xa}$.*

Lemma 21 (R-Promotion for Reflexivity) *Let $\Phi_{xa} \sim \Phi_{xd}$. If $\Phi_{xd} \vdash \mathsf{is\_tm}\ M$ then $\Phi_{xa} \vdash \mathsf{aeq}\ M\ M$.*

The proofs are analogous to Lemmas 15 and 16, with the proof of Lemma 21 requiring Lemma 20.

Theorem 22 (Completeness, R Version) *Let $\Phi_{xa} \sim \Phi_{xd}$. If $\Phi_{xd} \vdash \mathsf{deq}\ M\ N$ then $\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$.*

Proof By induction on the derivation $\mathcal{D} :: \Phi_{xd} \vdash \mathsf{deq}\ M\ N$.
Case:

$$\mathcal{D} = \dfrac{}{\Phi_{xd} \vdash \mathsf{deq}\ M\ M}\ de_r$$

$\Phi_{xd} \vdash \mathsf{is\_tm}\ M$ by (implicit) assumption
$\Phi_{xa} \vdash \mathsf{aeq}\ M\ M$ by Theorem 21

Case:

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1\\ \Phi_{xd} \vdash \mathsf{deq}\ M\ L\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Phi_{xd} \vdash \mathsf{deq}\ L\ N\end{array}}{\Phi_{xd} \vdash \mathsf{deq}\ M\ N}\ de_t$$

$\Phi_{xa} \vdash \mathsf{aeq}\ M\ L$ and $\Phi_{xa} \vdash \mathsf{aeq}\ L\ N$ by IH on $\mathcal{D}_1$ and $\mathcal{D}_2$
$\Phi_{xa} \vdash \mathsf{aeq}\ M\ N$ by Theorem 10 (2)

Case:

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}'\\ \Phi_{xd}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x \vdash \mathsf{deq}\ M\ N\end{array}}{\Phi_{xd} \vdash \mathsf{deq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)}\ de_l$$

$\Phi_{xa} \sim \Phi_{xd}$ by assumption
$\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \sim \Phi_{xd}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x$ by rule $crel_{ad}$
$\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M\ N$ by IH on $\mathcal{D}'$
$\Phi_{xa} \vdash \mathsf{aeq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)$ by rule ae_l

Only one promotion lemma is required in this proof, for the reflexivity case (which requires one occurrence each of *c-str* and *c-wk*), and no strengthening or weakening is needed in the lambda case (thus no occurrences of *d-str/wk* in this proof). In contrast, the proof of the G version of this theorem (Theorem 19) uses 3 occurrences of each of *c-str* and *c-wk* via promotion Lemma 18 and one occurrence each of *d-str* and *d-wk* in the lambda case.

3.4 Order

A consequence of viewing contexts as sequences is that *order* comes into play, and therefore the need to consider *exchanging* the elements of a context. This happens when, for example, a judgment singles out a particular occurrence of an assumption in head position. We exemplify this with a "parallel" substitution property for algorithmic equality, stated below. The proof also involves some slightly more sophisticated reasoning about names in the variable case than previously observed. Furthermore, note that this substitution property does not "come for free" in a HOAS encoding in the way, for example, that type substitution (Lemma 25) does.

Theorem 23 (Pairwise Substitution) *If* $\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M_1\ M_2$ *and* $\Phi_{xa} \vdash \mathsf{aeq}\ N_1\ N_2$*, then* $\Phi_{xa} \vdash \mathsf{aeq}\ ([N_1/x]M_1)\ ([N_2/x]M_2)$.

Proof By induction on the derivation $\mathcal{D} :: \Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M_1\ M_2$ and inversion on $\Phi_{xa} \vdash \mathsf{aeq}\ N_1\ N_2$. We show two cases.
Case:

$$\mathcal{D} = \frac{\mathsf{aeq}\ y\ y \in \Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x}{\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ y\ y}\ ae_v$$

We need to establish $\Phi_{xa} \vdash \mathsf{aeq}\ ([N_1/x]y)\ ([N_2/x]y)$.
Sub-case: $y = x$: Applying the substitution to the above judgment, we need to show $\Phi_{xa} \vdash \mathsf{aeq}\ N_1\ N_2$, which we have.
Sub-case: $\mathsf{aeq}\ y\ y \in \Phi_{xa}$, for $y \neq x$. Applying the substitution in this case gives us $\Phi_{xa} \vdash \mathsf{aeq}\ y\ y$, which we have by assumption.
Case:

$$\mathcal{D} = \frac{\overset{\mathcal{D}'}{\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x, \mathsf{is\_tm}\ y; \mathsf{aeq}\ y\ y \vdash \mathsf{aeq}\ M_1\ M_2}}{\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ (\mathsf{lam}\, y.\, M_1)\ (\mathsf{lam}\, y.\, M_2)}\ de_l$$

$\Phi_{xa}, \mathsf{is\_tm}\ y; \mathsf{aeq}\ y\ y, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M_1\ M_2$ by *exc* on $\mathcal{D}'$
$\Phi_{xa} \vdash \mathsf{aeq}\ N_1\ N_2$, by assumption
$\Phi_{xa}, \mathsf{is\_tm}\ y; \mathsf{aeq}\ y\ y \vdash \mathsf{aeq}\ N_1\ N_2$ by *d-wk*
$\Phi_{xa}, \mathsf{is\_tm}\ y; \mathsf{aeq}\ y\ y \vdash \mathsf{aeq}\ ([N_1/x]M_1)\ ([N_2/x]M_2)$ by IH
$\Phi_{xa} \vdash \mathsf{aeq}\ [N_1/x](\mathsf{lam}\, y.\, M_1)\ [N_2/x](\mathsf{lam}\, y.\, M_2)$ by rule ae_l and possible renaming.

We remark that there are more general ways to formulate properties such as Theorem 23 that do *not* require (on paper) exchange; for example,

> If $\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x, \Phi'_{xa} \vdash \mathsf{aeq}\ M_1\ M_2$ and $\Phi_{xa} \vdash \mathsf{aeq}\ N_1\ N_2$, then $\Phi_{xa}, \Phi'_{xa} \vdash \mathsf{aeq}\ ([N_1/x]M_1)\ ([N_2/x]M_2)$.

The proof of the latter statement has a similar structure to the previous one, except that it uses *d-wk* in the first variable sub-case, while the binding case does not employ any structural property to apply the induction hypothesis, by taking $(\Phi''_{xa}, \mathsf{is\_tm}\ y; \mathsf{aeq}\ y\ y)$ as Φ'_{xa}. While this works well in a paper and pencil style, it is much harder to mechanize, since it brings in reasoning about appending and splitting lists that are foreign to the matter at hand.

We conclude by noting that there are examples where exchange cannot be applied, since the dependency proviso is not satisfied. Cases in point are substitution lemmas for dependent types. Here, other encoding techniques must be used, as explored in Crary (2009).

3.5 Uniqueness

Uniqueness of context variables plays an unsurprisingly important role in proving type uniqueness, i.e. every lambda-term has a unique type. For the sake of this discussion it is enough to consider the monomorphic case, where abstractions include type annotations on bound variables, and types consist only of a ground type and a function arrow.

$$\begin{array}{l} \text{Terms } M ::= y \mid \mathsf{lam}\, x^A.\, M \mid \mathsf{app}\ M_1\ M_2 \\ \text{Types } A\ ::= \mathsf{i} \mid \mathsf{arr} A\, B \end{array}$$

The typing rules are the obvious subset of the ones presented in Sect. 2, yielding:

$$\text{Context Schema } S_t := \mathsf{is\_tm}\ x; x{:}A$$

The statement of the theorem requires only a single context and thus there is no distinction to be made between the R and G versions.

Theorem 24 (Type Uniqueness) *If $\Phi_t \vdash M : A$ and $\Phi_t \vdash M : B$ then $A = B$.*

Proof The proof is by induction on the first derivation and inversion on the second. We show only the variable case where uniqueness plays a central role.
Case:

$$\mathcal{D} = \frac{x{:}A \in \Phi_t}{\Phi_t \vdash x : A}\ of_v$$

We know that $x{:}A \in \Phi_t$ by rule of_v. By definition, Φ_t contains block $(\mathsf{is\_tm}\ x; x{:}A)$. Moreover, we know $\Phi_t \vdash x : B$ by assumption. By inversion using rule of_v, we know that $x{:}B \in \Phi_t$, which means that Φ_t contains block $(\mathsf{is\_tm}\ x; x{:}B)$. Since all assumptions about x occur uniquely, these must be the same block. Thus A must be identical to B.

3.6 Substitution

In this section we address the interaction of the substitution property with context reasoning. It is well known and rightly advertised that substitution lemmas come "for free" in HOAS encodings, since substitutivity is just a by-product of hypothetical-parametric judgments. We refer to Pfenning (2001) for more details. A classic example is the proof of type preservation for a functional programming language, where a lemma stating that substitution preserves typing is required in every case that involves a β-reduction. However, this example theorem is unduly restrictive since functional programs are closed expressions; in fact, the proof proceeds by induction on (closed) evaluation and inversion on typing, hence only addressing contexts in a marginal way. We thus discuss a similar proof for an evaluation relation that "goes under a lambda" and we choose parallel reduction, as it is a standard relation also used in other important case studies such as the Church-Rosser theorem. The context schema and relevant rules are below.

Parallel Reduction

$$\frac{x \rightsquigarrow x \in \Gamma}{\Gamma \vdash x \rightsquigarrow x}\ pr_v \qquad \frac{\Gamma, \mathsf{is\_tm}\ x; x \rightsquigarrow x \vdash M \rightsquigarrow N}{\Gamma \vdash \mathsf{lam}\, x.\, M \rightsquigarrow \mathsf{lam}\, x.\, N}\ pr_l$$

$$\frac{\Gamma, \mathsf{is\_tm}\ x; x \rightsquigarrow x \vdash M \rightsquigarrow M' \qquad \Gamma \vdash N \rightsquigarrow N'}{\Gamma \vdash (\mathsf{app}\ (\mathsf{lam}\, x.\, M)\ N) \rightsquigarrow [N'/x]M'}\ pr_\beta$$

$$\frac{\Gamma \vdash M \rightsquigarrow M' \qquad \Gamma \vdash N \rightsquigarrow N'}{\Gamma \vdash (\mathsf{app}\ M\ N) \rightsquigarrow (\mathsf{app}\ M'\ N')}\ pr_a$$

$$\text{Context Schema } S_r := \mathsf{is\_tm}\ x; x \rightsquigarrow x$$

The relevant substitution lemma is:

Lemma 25 *If* $\Phi_t, \mathsf{is\_tm}\ x; x{:}A \vdash M : B$ *and* $\Phi_t \vdash N : A$*, then* $\Phi_t \vdash [N/x]M : B$*.*

Proof While this is usually proved by induction on the first derivation, we show it as a corollary of the substitution principles.

$\Phi_t, \mathsf{is\_tm}\ x; x{:}A \vdash M : B$	by assumption
$\Phi_t, \mathsf{is\_tm}\ N; N{:}A \vdash [N/x]M : B$	by parametric substitution
$\Phi_t, \mathsf{is\_tm}\ N \vdash [N/x]M : B$	by hypothetical substitution
$\Phi_t \vdash \mathsf{is\_tm}\ N$	by (implicit) assumption
$\Phi_t \vdash [N/x]M : B$	by hypothetical substitution

We show only the R version of type preservation. For the G version, the context schema is obtained by combining the schemas S_r and S_t similarly to how S_{da} was defined to combine S_{xa} and S_{xd} in Sect. 3.3.1. We leave it to the reader to complete such a proof. For the R version, we introduce the customary context relation, which in this case is:

$$\frac{\Phi_r \sim \Phi_t}{\Phi_r, \mathsf{is\_tm}\ x; x \rightsquigarrow x \sim \Phi_t, \mathsf{is\_tm}\ x; x{:}A}\ \mathit{crel}_{rt}$$

Theorem 26 (Type Preservation for Parallel Reduction) *Assume* $\Phi_r \sim \Phi_t$*. If* $\Phi_r \vdash M \rightsquigarrow N$ *and* $\Phi_t \vdash M : A$*, then* $\Phi_t \vdash N : A$*.*

Proof The proof is by induction on the derivation $\mathcal{D} :: \Phi_r \vdash M \rightsquigarrow N$ and inversion on $\Phi_t \vdash M : A$. We show only two cases:

Case:

$$\mathcal{D} = \frac{x \rightsquigarrow x \in \Phi_r}{\Phi_r \vdash x \rightsquigarrow x}\ pr_v$$

We know that in this case $M = x = N$. Then the result follows trivially.

Case:

$$\mathcal{D} = \frac{\begin{matrix}\mathcal{D}_1 \\ \Phi_r, \mathsf{is\_tm}\ x; x \rightsquigarrow x \vdash M \rightsquigarrow M'\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Phi_r \vdash N \rightsquigarrow N'\end{matrix}}{\Phi_r \vdash (\mathsf{app}\ (\mathsf{lam}\, x.\, M)\ N) \rightsquigarrow [N'/x]M'}\ pr_\beta$$

$\Phi_t \vdash (\mathsf{app}\ (\mathsf{lam}\, x.\, M)\ N) : A$	by assumption
$\Phi_t \vdash (\mathsf{lam}\, x.\, M) : \mathsf{arr} B\ A$ and $\Phi_t \vdash N : B$	by inversion on rule of_a
$\Phi_t \vdash N' : B$	by IH on $\mathcal{D}_2$ and the latter
$\Phi_t, \mathsf{is\_tm}\ x; x{:}B \vdash M : A$	by inversion on rule of_l
$\Phi_r \sim \Phi_t$	by assumption
$(\Phi_r, \mathsf{is\_tm}\ x; x \rightsquigarrow x) \sim (\Phi_t, \mathsf{is\_tm}\ x; x{:}B)$	by rule crel_{rt}
$\Phi_t, \mathsf{is\_tm}\ x; x{:}B \vdash M' : A$	by IH
$\Phi_t \vdash [N'/x]M' : A$	by Lemma 25 (substitution)

If we were to prove a similar result for the polymorphic λ-calculus, we would need another substitution lemma, namely:

Lemma 27 *If* $\Phi_{\alpha t}, \mathsf{is\_tp}\ \alpha \vdash M : B$ *and* $\Phi_{\alpha t} \vdash \mathsf{is\_tp}\ A$*, then* $\Phi_{\alpha t} \vdash [A/\alpha]M : [A/\alpha]B$*.*

Again, this follows immediately from parametric and hypothetical substitution, whereas a direct inductive proof may not be completely trivial to mechanize.

4 The ORBI Specification Language

ORBI (Open challenge problem Repository for systems supporting reasoning with BInders) is an open repository for sharing benchmark problems based on the notation we have developed. 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 the 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 systems.

The desire for ORBI to cater to both type and proof theoretic frameworks requires an almost impossible balancing act between the two views. While all the systems we plan to target are essentially two-level, they differ substantially, as we will see in much more detail in the companion paper (Felty et al, 2014). 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 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[6] tools.

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 this paper. 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. 6.

4.1 Problem Description

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 type checked by Beluga thereby eliminating many syntactically correct but meaningless expressions.

Object languages are written according to the EBNF (Extended Backus-Naur Form) grammar in Fig. 2, which uses certain 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

[6] Following TPTP's nomenclature (Sutcliffe, 2009), we call "ORBI2X" any tool taking an ORBI specification as input; for example, the translator for Hybrid mentioned earlier translates syntax, inference rules, and context definitions of ORBI into input to the Coq version of Hybrid, and is designed so that it can be adapted fairly directly to output Abella scripts.

```
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}
```

Fig. 2 ORBI Grammar for Syntax, Judgments, Inference Rules, and Context Schemas

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 which 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 used in Subsections 3.1, 3.3, and 3.4. The full ORBI specification is given in Appendix B, and all examples described in this section are taken from that specification. For the remaining example specifications, we refer the reader to the the companion paper (Felty et al, 2014) or to `https://github.com/pientka/ORBI`.

To assist compact translations to systems that do not include the LF language, we also support *directives* written as comments of a special form, i.e., they are prefixed by `%` and ignored by the LF type checker. For example, we provide directives that allow us to distinguish between the syntax definition of an object language and the definition of its judgments and inference rules. (See Appendix B.) Directives, including their grammar, are detailed in Sect. 4.2.

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 declaration `tm:type`. This declaration along with those of the constructors `app` and `lam` in the `Syntax` section 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.[7] In our running example, we have two judgments, `aeq` and `deq` of type `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).
```

$$\frac{\mathsf{aeq}\ M_1\ N_1 \qquad \mathsf{aeq}\ M_2\ N_2}{\mathsf{aeq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}\ ae_a$$

Uppercase letters such as `M1` denote schematic variables, which are implicitly quantified at the outermost level, namely `{M1:tm}`, as 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)).
```

$$\dfrac{\begin{array}{c}\overline{\mathsf{is\_tm}\ x}\ x \qquad \overline{\mathsf{aeq}\ x\ x}\ ae_v \\ \vdots \\ \mathsf{aeq}\ M\ N\end{array}}{\mathsf{aeq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)}\ ae_l^{x,ae_v}$$

We view the $\mathsf{is\_tm}\ x$ assumption as the parametric assumption `x:tm`, while the hypothesis $\mathsf{aeq}\ x\ x$ (and its scoping) is encoded within the embedded implication `aeq x x` $\to$ `aeq (M x) (N x)` in the current (informal) signature augmented with the dynamic declaration for `x`.[8] Recall that the "variable" case of an implicit-context presentation, namely ae_v, is folded inside the binder case.

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, schema S_{xa} in Sect. 3.1.2 appears in the `Schemas` section of Appendix B as:

```
schema xaG: block (x:tm; u:aeq x x).
```

As another example, in this case illustrating a schema sporting alternatives, we encode the schema S_{aeq} from polymorphic equality as:

```
schema aeqG: block (a:tp; u:atp a a) + block (x:tm; v:aeq x x).
```

While we can 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.

[7] There are several excellent tutorials (Pfenning, 2001; Harper and Licata, 2007) on how to encode OLs in LF, and hence we keep it brief.

[8] As is well known, parametric assumptions and embedded implication are unified in the type-theoretic view.

Definitions So far we have considered the specification language for encoding formal systems. ORBI also supports declaring inductive definitions for specifying context relations and theorems. We start with the grammar for inductive definition (Fig. 3). An inductive predicate is given a `r_kind` by the production `def_dec`. 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 a given schema. Hence the base predicate is of the form `id {ctx}` relating different contexts.

```
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      ::= nil | id | ctx "," blk
```

Fig. 3 ORBI Grammar for Inductive Definitions describing Context Relations

For example, the relation $\Phi_x \sim \Phi_{xa}$ is encoded in the `Definitions` section of Appendix B as:

```
inductive xaR : {G:xG} {H:xaG} prop =
| xa_nil: xaR nil nil
| xa_cons: xaR G H -> xaR (G, block x:tm) (H, 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; this also allows for a straightforward translation. Twelf's type theoretic framework, however, is not rich enough to support inductive definitions.

4.2 Language for Theorems and Directives

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. 4. 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 *)
```

Fig. 4 ORBI Grammar for Theorems

The specification of the G and R versions of the completeness theorem is as follows:

```
theorem ceqG: {G:daG} [G |- deq M N] -> [G |- aeq M N].
theorem ceqR: {G:xdG}{H:xaG} daR G H -> [G |- deq M N] -> [H |- aeq M N].
```

This and all the others theorems pertaining to the development of the meta-theory of algorithmic and declarative equality are listed in the `Theorems` section of Appendix B. The theorems stated are a straightforward encoding of the main theorems in Subsections 3.1, 3.3, and 3.4.

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. hypthetical 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 As we have mentioned before, *directives* are comments that help the ORBI2X tools to generate target representations of the ORBI specifications. The idea is reminiscent of what *Ott* (Sewell et al, 2010) 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. 5.

```
dir       ::=  '%' sy_id what 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
```

Fig. 5 ORBI Grammar for Directives

Most of the directives that we consider in this version of ORBI are dedicated to help the translations into proof-theoretical systems, although we include also 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. Beyond directives (`sepr`) meant to structure ORBI specs, the instructions `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 first line in the `Directives` section of Appendix B states the directive "`% [hy,ab] wf tm`" that 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 Abella and Hybrid:

$$\forall M, N.\ (\forall x.\ \mathsf{is\_tm}\ x \to \mathsf{aeq}\ x\ x \to \mathsf{aeq}\ Mx\ Nx) \to \mathsf{aeq}\ (\mathsf{lam}\ M)\ (\mathsf{lam}\ N)$$
$$\forall M, N.\ (\forall x.\ \mathsf{aeq}\ x\ x \to \mathsf{aeq}\ Mx\ Nx) \to \mathsf{aeq}\ (\mathsf{lam}\ M)\ (\mathsf{lam}\ N)$$

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. In fact, in the previous section, we assumed 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 in de_l.
% [hy,ab] explicit M in de_r.
```

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

$$\forall M, N.\ (\forall x.\ \mathsf{is\_tm}\ x \to \mathsf{deq}\ x\ x \to \mathsf{deq}\ Mx\ Nx) \to \mathsf{deq}\ (\mathsf{lam}\ M)\ (\mathsf{lam}\ N)$$
$$\forall M.\ \mathsf{is\_tm}\ M \to \mathsf{aeq}\ M\ M$$

The case for schemas is analogous: in the proof-theoretic view, schemas are translated to unary inductive predicates. Again, typing information is left implicit in the translation unless a directive is included. For example, the `xaG` schema with no associated directive will be translated to a definition that expresses that whenever context `G` has schema `xaG`, then so does `G,aeq x x`. For the `daG` schema, with directive

```
% [hy,ab] explicit x in daG.
```

the translation will express that whenever `G` has schema `daG`, then so does `G, (is_tm x;deq x x;aeq x x)`.

Similarly, directives in context relations, such as:

```
% [hy,ab] explicit x in G in xaR.
```

also state which well-formedness annotations to make explicit in the translated version. In this case, when translating the definition of `xaR` 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. (Speculative directives are omitted from Appendix B). For example, theorems `reflG` is proven by induction over `M`. As a consequence, `M` must be explicit.

```
% [hy,ab,bel] explicit M in H in reflG.
```

Hybrid and Abella interpret the directive by adding an explicit assumption $[H \vdash \mathsf{is\_tm}\ M]$, as illustrated by the result of the translation:

$$\forall H, M.\ [H \vdash \mathsf{is\_tm}\ M] \to [H \vdash \mathsf{aeq}\ M\ M]$$

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 (denoted by `..`). In short, the directive allows us to lift the type specified in ORBI to a contextual type which 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 in H in symG.
% [bel] implicit N in H in symG.
```

4.3 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 not to put 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. 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.

5 Related Work

Our approach to structuring contexts of assumptions takes its inspiration from Martin-Löf's theory of judgments (Martin-Löf, 1996), especially in the way it has been realized in Edinburgh LF (Harper et al, 1993). 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 latter was introduced in Schürmann (2000), and used in Schürmann and Pfenning (2003) for the meta-theory of Twelf and in Momigliano (2000) for different purposes. It was further explicated in Harper and Licata (2007)'s review of Twelf's methodology, but its treatment remained unsatisfactory since the notion of worlds is extra-logical. Recent work (Wang and Nadathur, 2013) on a logical rendering of Twelf's totality checking has so far been limited to closed objects.

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* (Sutcliffe, 2009), 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* (Benzmüller et al, 2008), in particular in its construction in stages.

The success of TPTP has spurned other benchmark suites in related subjects, see for example *SATLIB* (Hoos and Stützle, 2000); 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 both Twelf's wiki (`http://twelf.org/wiki/Case_studies`), Abella's library (`http://abella-prover.org/examples`) and Beluga's distribution contain a set of context-intensive examples, some of which coincide with the ones presented here. As such they are prime candidates to be included in ORBI.

Other projects have put forward LF as a common ground: *Logosphere*'s goal (`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* (Aldrich et al, 2008) 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.

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 (Filliâtre, 2013), whose specifications are then translated in 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, such as 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 (Sewell et al, 2010) 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 notion 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* (Felleisen et al, 2009) and also the K framework (Roşu and Şerbănuţă, 2010). 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 carried out.

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.

6 Conclusion and Future Work

We have presented an initial set of benchmarks that highlight a variety of different aspects of reasoning within a context of assumptions. We have also provided an infrastructure for formalizing these benchmarks in a variety of HOAS-based systems, and for facilitating their comparison. We have developed a framework for expressing contexts of assumptions as structured sequences, which provides additional structure to contexts via schemas and characterizes their basic properties. Finally, we have designed (the initial version of) the ORBI (Open challenge problem Repository for systems supporting reasoning with BInders) specification language, and created an open repository of specifications, which initially contains the benchmarks introduced in this paper.

Selecting a small set of benchmarks has an inherent element of arbitrariness. The reader may complain that there are many other features and issues not covered in Sect. 3. We agree and we mention some additional categories, which we could not discuss in the present paper for the sake of space, but which will (eventually) make it into the ORBI repository:

- One of the weak spots of most current HOAS-based systems is the lack of libraries, built-in data-types and related decision procedures: for example, case studies involving calculi of explicit substitutions require a small corpus of arithmetic facts, that, albeit trivial, still need to be (re)proven, while they could be automatically discharged by decision procedures such as Coq's `omega`.[9]
- There are also specifications that are *functional* in nature, such as those that descend through the structure of a lambda term, say counting its depth, the number of bound occurrences of a given variable etc.; most HOAS systems would encode those functions relationally, but this entails again the additional proof obligations of proving those relations total and deterministic.
- In the benchmarks that we have presented all blocks are composed of *atoms*, but there are natural specifications, to wit the solution to the PoplMark challenge in Pientka (2007), where contexts have more structure, as they are induced by *third-order* specifications. For example, the rule for subtyping universally quantified types introduces a non-atomic assumption about transitivity, of the form:

  ```
  {a:tp}({U:tp} {V:tp} sub a U → sub U V → sub a V).
  ```
- Proofs by *logical relations* typically require, in order to define reducibility candidates, inductive definitions and strong function spaces, i.e., a function space that does not only model binding. A direct encodings of those proofs is out of reach for systems such as Twelf, although indirect encodings exist (Schürmann and Sarnat, 2008). Other systems, such as Beluga and Abella, are well capable of encoding such proofs, but differ in how this is accomplished, see Cave and Pientka (2014) and Gacek et al (2012).
- Finally, a subject that is gaining importance is the encoding of *infinite* behavior, typically realized via some form of *co-induction*. Context-intensive case studies have been explored for example in Momigliano (2012).

One of the outcomes of our framework for expressing contexts of assumptions is the unified treatment of all weakening/strengthening/exchange re-arrangements, via the `rm` and `perm` operations. This opens the road to a *lattice-theoretic* view of declarations and contexts, where, roughly, $x \preceq y$ holds iff x can be reached from y

[9] Case in point, the strong normalization proof for the $\lambda\sigma$ calculus in Abella, see `http://abella-prover.org/examples/lambda-calculus/exsub-sn/`, 15% of which consists of basic facts about addition.

by some rm operation: a generalized context will be the join of two contexts and context relations can be identified by navigating the lattice starting from the join of the to-be-related contexts. We plan to develop this view and use it to convert G proofs into R and vice versa, as a crucial step towards breaking the proof/type theory barrier.

The description of ORBI given in Sect. 4 is best thought of as a stepping stone towards a more comprehensive specification language, much as *THF0* (Benzmüller et al, 2008) 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 Cave and Pientka (2012) and Gacek et al (2012) 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 a notion of proof 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 specification such nominal, locally nameless etc.

Acknowledgements The first and third author acknowledge the support of the Natural Sciences and Engineering Research Council of Canada. We thank Kaustuv Chaudhuri, Andrew Gacek, Nada Habli, and Dale Miller for discussing some aspects of this work with us. The first author would also like to extend her gratitude to the University of Ottawa's Women's Writers Retreats.

References

Aldrich J, Simmons RJ, Shin K (2008) SASyLF: An educational proof assistant for language theory. In: 2008 International Workshop on Functional and Declarative Programming in Education, ACM, pp 31–40

Aydemir BE, Bohannon A, Fairbairn M, Foster JN, Pierce BC, Sewell P, Vytiniotis D, Washburn G, Weirich S, Zdancewic S (2005) Mechanized metatheory for the masses: The POPLMARK challenge. In: Eighteenth International Conference on Theorem Proving in Higher Order Logics, Springer, LNCS, vol 3603, pp 50–65

Benzmüller C, Rabe F, Sutcliffe G (2008) THF0—the core of the TPTP language for higher-order logic. In: Fourth International Joint Conference on Automated Reasoning, Springer, LNCS, vol 5195, pp 491–506

Cave A, Pientka B (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

Cave A, Pientka B (2014) Mechanizing logical relation proofs using contextual objects. Tech. rep., School of Computer Science, McGill

Crary K (2009) Explicit contexts in LF (extended abstract). In: Third International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP 2008, ENTCS, vol 228, pp 53–68

Felleisen M, Findler RB, Flatt M (2009) Semantics Engineering with PLT Redex. The MIT Press

Felty A, Pientka B (2010) Reasoning with higher-order abstract syntax and contexts: A comparison. In: First International Conference on Interactive Theorem Proving, Springer, LNCS, vol 6172, pp 227–242

Felty A, Momigliano A, Pientka B (2014) The next 700 challenge problems for reasoning with higher-order abstract syntax representations: Part 2—a survey, submitted for publication simultaneously with this paper

Felty AP, Momigliano A (2012) Hybrid: A definitional two-level approach to reasoning with higher-order abstract syntax. Journal of Automated Reasoning 48(1):43–105

Fernández M, Urban C (2012) Preface: Theory and applications of abstraction, substitution and naming. Journal of Automated Reasoning 49(2):111–114

Filliâtre JC (2013) One logic to use them all. In: 24th International Conference on Automated Deduction, Springer, LNCS, vol 7898, pp 1–20

Gacek A (2008) The Abella interactive theorem prover (system description). In: Fourth International Joint Conference on Automated Reasoning, Springer, LNCS, vol 5195, pp 154–161

Gacek A, Miller D, Nadathur G (2012) A two-level logic approach to reasoning about computations. Journal of Automated Reasoning 49(2):241–273

Girard JY, Lafont Y, Tayor P (1990) Proofs and Types. Cambridge University Press

Habli N, Felty AP (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, vol 14, pp 67–76

Harper R, Licata DR (2007) Mechanizing metatheory in a logical framework. Journal of Functional Programming 17(4-5):613–673

Harper R, Honsell F, Plotkin G (1993) A framework for defining logics. Journal of the Association for Computing Machinery 40(1):143–184

Hoos HH, Stützle T (2000) Satlib: An online resource for research on SAT. In: Gent I, Maaren HV, Walsh T (eds) SAT 2000: Highlights of Satisfiability Research in the Year 2000, IOS Press, Frontiers in Artificial Intelligence and Applications, vol 63, pp 283–292

Martin-Löf P (1996) On the meanings of the logical constants and the justifications of the logical laws. Nordic Journal of Philosophical Logic 1(1):11–60

Miller D (1991) A logic programming language with lambda-abstraction, function variables, and simple unification. Journal of Logic and Computation 1(4):497–536

Miller D, Palamidessi C (1999) Foundational aspects of syntax. ACM Computing Surveys 31(3es):1–6, Article No. 11

Momigliano A (2000) Elimination of negation in a logical framework. In: Computer Sceince Logic, Springer, LNCS, vol 1862, pp 411–426

Momigliano A (2012) A supposedly fun thing I may have to do again: A HOAS encoding of Howe's method. In: Seventh ACM Workshop on Logical Frameworks and Meta-Languages, Theory and Practice, ACM Press, pp 33–42

Momigliano A, Martin AJ, Felty AP (2008) Two-level Hybrid: A system for reasoning using higher-order abstract syntax. In: Second Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, ENTCS, vol 196, pp 85–93

Pfenning F (2001) Computation and deduction, URL `http://www.cs.cmu.edu/~fp/courses/comp-ded/handouts/cd.pdf`, accessed 14 October 2014

Pientka B (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, Springer, LNCS, pp 246–261

Pientka B (2008) A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In: Thirty-Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, pp 371–382

Pientka B, Dunfield J (2010) Beluga: A framework for programming and reasoning with deductive systems (system description). In: Fifth International Joint Conference on Automated Reasoning, Springer, LNCS, vol 6173, pp 15–21

Pierce BC (2002) Types and Programming Languages. MIT Press

Pierce BC, Weirich S (2012) Preface to special issue: The POPLMARK challenge. Journal of Automated Reasoning 49(3):301–302

Poswolsky AB, Schürmann C (2008) Practical programming with higher-order encodings and dependent types. In: Seventeenth European Symposium on Programming, Springer, LNCS, vol 4960, pp 93–107

Roşu G, Şerbănuţă TF (2010) An overview of the K semantic framework. Journal of Logic and Algebraic Programming 79(6):397–434

Schürmann C (2000) Automating the meta theory of deductive systems. PhD thesis, Department of Computer Science, Carnegie Mellon University, available as Technical Report CMU-CS-00-146

Schürmann C (2009) The Twelf proof assistant. In: 22nd International Conference on Theorem Proving in Higher Order Logics, Springer, LNCS, vol 5674, pp 79–83

Schürmann C, Pfenning F (2003) A coverage checking algorithm for LF. In: Sixteenth International Conference on Theorem Proving in Higher Order Logics, Springer, LNCS, vol 2758, pp 120–135

Schürmann C, Sarnat J (2008) Structural logical relations. In: Twenty-Third Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society, pp 69–80

Sewell P, Nardelli FZ, Owens S, Peskine G, Ridge T, Sarkar S, Strniša R (2010) Ott: Effective tool support for the working semanticist. Journal of Functional Programming 20(1):71–122

Sutcliffe G (2009) The TPTP problem library and associated infrastructure. Journal of Automated Reasoning 43(4):337–362

Wang Y, Nadathur G (2013) Towards extracting explicit proofs from totality checking in Twelf. In: Eighth ACM SIGPLAN International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, ACM Press, pp 55–66

A Overview of Benchmarks

In this appendix, we provide a quick reference guide to some of the key elements of the benchmark problems discussed in Section 3. In the tables below, ULC (STLC) stands for the untyped (simply-typed) lambda-calculus, and POLY stands for the polymorphic lambda calculus. The entry "same" means that there is no difference between the R and G version of the theorem because there is only one context involved.

A.1 A Recap of Benchmark Theorems

Theorem	Thm No.	Version	Page
aeq-reflexivity for ULC	7	R	16
aeq-reflexivity for ULC	8	G	17
aeq-symmetry and transitivity for ULC	10	same	18
atp-reflexivity for POLY	11	G	19
aeq-reflexivity for POLY	13	G	19
atp-reflexivity for POLY	14	R	20
aeq-reflexivity for POLY	17	R	21
aeq/deq-completeness for ULC	19	G	22
aeq/deq-completeness for ULC	22	R	23
type uniqueness for STLC	24	same	26
type preservation for parallel reduction for STLC	26	R	27
aeq-parallel substitution for ULC	23	same	24

A.2 A Recap of Schemas and Their Usage

Context	Schema	Block	Description/Used in:
Φ_α	S_α	$\mathsf{is\_tp}\ \alpha$	type variables
Φ_x	S_x	$\mathsf{is\_tm}\ x$	term variables
$\Phi_{\alpha x}$	$S_{\alpha x}$	$\mathsf{is\_tp}\ \alpha + \mathsf{is\_tm}\ x$	type/term variables
$\Phi_{\alpha t}$	$S_{\alpha t}$	$\mathsf{is\_tp}\ \alpha + \mathsf{is\_tm}\ x; x{:}T$	type-checking for POLY
Φ_{xa}	S_{xa}	$\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x$	Thm 8, 10, and 23
Φ_{atp}	S_{atp}	$\mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha$	Thm 11
Φ_{aeq}	S_{aeq}	$\mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha\ +\ \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x$	Thm 13
Γ_{da}	S_{da}	$\mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x; \mathsf{aeq}\ x\ x$	Thm 19
Φ_{xd}	S_{xd}	$\mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x$	Thm 22
Φ_t	S_t	$\mathsf{is\_tm}\ x; \mathtt{oft}\ x\ A$	Thm 24
Φ_r	S_r	$\mathsf{is\_tm}\ x; x \rightsquigarrow x$	Thm 26

A.3 A Recap of the Main Context Relations and Their Usage

Relation	Related Blocks	Used in:
$\Phi_x \sim \Phi_{xa}$	$\mathsf{is\_tm}\ x \sim (\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x)$	Thm 7
$\Phi_\alpha \sim \Phi_{atp}$	$\mathsf{is\_tp}\ \alpha \sim (\mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha)$	Thm 14
$\Phi_{\alpha x} \sim \Phi_{aeq}$	$\Phi_x \sim \Phi_{xa}$ plus $\Phi_\alpha \sim \Phi_{atp}$	Thm 17
$\Phi_{xa} \sim \Phi_{xd}$	$(\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x) \sim (\mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x)$	Thm 22
$\Phi_r \sim \Phi_t$	$(\mathsf{is\_tm}\ x; x \rightsquigarrow x) \sim (\mathsf{is\_tm}\ x; x{:}A)$	Thm 26

B ORBI Specification of Algorithmic and Declarative Equality

The following ORBI specification provides a complete encoding of the example of algorithmic vs. declarative equality used in Subsections 3.1, 3.3, and 3.4.

```
%% Syntax
tm: type.

app: tm -> tm -> tm.
lam: (tm -> tm) -> tm.
```

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

%% Rules
ae_a: aeq M1 N1 -> aeq M2 N2 -> aeq (app M1 M2) (app N1 N2).
ae_l: ({x:tm} aeq x x -> aeq (M x) (N x))
        -> aeq (lam (\x. M x)) (lam (\x. N x)).

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 M1 M2 -> deq M2 M1.
de_t: deq M1 M2 -> deq M2 M3 -> deq M1 M3.

%% 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).

%% Definitions
inductive xaR : {G:xG} {H:xaG} prop =
| xa_nil: xaR nil nil
| xa_cons: xaR G H -> xaR (G, block x:tm) (H, block x:tm; u:aeq x x).

inductive daR : {G:xdG} {H:xaG} prop =
| da_nil: daR nil nil
| da_cons: daR G H -> daR (G, block x:tm; v:deq x x)
                          (H, block x:tm; u:aeq x x).

%% Theorems
theorem reflG: {H:xaG} {M:tm} [H |- aeq M M].
theorem symG: {H:xaG}{M:tm}{N:tm} [H |- aeq M N] -> [H |- aeq N M].
theorem transG: {H:xaG}{M:tm}{N:tm}{L:tm}
              [H |- aeq M N] & [H |- aeq N L] -> [H |- aeq M L].
theorem ceqG: {G:daG} [G |- deq M N] -> [G |- aeq M N].
theorem substG: {H:xaG}{M1:tm->tm}{M2:tm}{N1:tm}{N2:tm}
  [H, block x:tm; aeq x x |- aeq (M1 x) (M2 x)] & [H |- aeq N1 N2] ->
  [H |- aeq (M1 N1) (M2 N2)].

theorem reflR : {G:xG}{H:xaG}{M:tm} xaR G H -> [H |- aeq M M].
theorem ceqR: {G:xdG}{H:xaG} daR G H -> [G |- deq M N] -> [H |- aeq M N].

%% Directives
% [hy,ab] wf tm.
% [hy,ab] explicit x in de_l.
% [hy,ab] explicit M in de_r.
% [hy,ab] explicit x in xG.
% [hy,ab] explicit x in xdG.
% [hy,ab] explicit x in daG.
% [hy,ab] explicit x in G in xaR.
% [hy,ab] explicit x in G in daR.
```