Math. Struct. in Comp. Science (2018), *vol.* 28, *pp.* 1507–1540.
doi:10.1017/S0960129517000093 First published online 5 May 2017

Benchmarks for reasoning with syntax trees containing binders and contexts of assumptions

AMY FELTY[†], ALBERTO MOMIGLIANO[‡] and
BRIGITTE PIENTKA[§]

[†]*School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada*
Email: afelty@eecs.uottawa.ca
[‡]*Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy*
Email: alberto.momigliano@unimi.it
[§]*School of Computer Science, McGill University, Montreal, Canada*
Email: bpientka@cs.mcgill.ca

Received 29 October 2015; revised 26 October 2016

A variety of logical frameworks supports the use of higher order abstract syntax 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 two-fold: (1) We develop a common infrastructure and language for describing benchmarks for systems supporting reasoning with binders, and (2) we present several concrete benchmarks, which highlight a variety of different aspects of reasoning within a context of assumptions. Our work provides the background for the qualitative comparison of different systems that we have completed in a separate paper. It also allows us to outline future fundamental research questions regarding the design and implementation of meta-reasoning systems.

1. Introduction

Ten years ago, the PoplMark challenge (Aydemir *et al.* 2005) stimulated considerable interest in mechanizing the meta-theory of programming languages, 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 he/she 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 low-level lemma that is used pervasively (and sometimes 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 yet 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 endeavour. 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 Cave 2015; Pientka and Dunfield 2010) and Delphin (Poswolsky and Schürmann 2008). All these systems also provide, to 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 modelled 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 (Felty and Momigliano 2012; Momigliano *et al.* 2008) as implemented on top of Coq and Isabelle/HOL, and the Abella system (Gacek 2008).

The contributions of the present paper are as follows: We develop a common framework and infrastructure for representing and describing *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. Using our common framework and language, we develop the proofs for these challenge problems in a systematic way. This provides a general footprint for mechanizing these examples, as we will see later. In a related paper (Felty *et al.* 2015a), we have carried out such a mechanization and comparison in four systems: Twelf, Beluga, Hybrid and Abella. The common framework we present here was key for the systematic comparison of these systems and understanding the trade-offs between them. It also may be seen as a first step towards developing a formal translation between different foundations, e.g., a translation between Beluga's type-theoretic foundation and the proof-theory underlying systems such as Hybrid or Abella.

We have also started an open repository of benchmarks called ORBI (Open challenge problem Repository for systems supporting reasoning with BInders), described in Felty *et al.* (2015b). ORBI includes a language for presenting benchmarks based on the common framework that we develop in this paper.

Challenge problems are important as they serve as an excellent regression suite and provide the basis for highlighting differences between and strengths and limitations of various systems. The problems described here can be viewed as an initial set. We 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. A solution to the proposed benchmarks should include an adequate way to represent syntax, contexts and judgment and a mechanized proof of those theorems.

The paper is structured as follows: In Section 2, we motivate our definition of contexts as 'structured sequences' that 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 Section 3. We conclude in Section 4 discussing related and future work. Appendix 4 provides a quick reference guide to the benchmarks. Full details about the challenge problems and their mechanization can be found at `https://github.com/pientka/ORBI`. The notation as well as the mechanization of these benchmarks in the four systems mentioned above are described in separate papers, Felty *et al.* (2015b) and Felty *et al.* (2015a), respectively.

2. Contexts of assumptions: Preliminaries and terminology

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 not to force upon us one particular foundation, so as to make our benchmarks more accessible to a wider audience.

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.

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

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 not only 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: $\mathsf{is\_tp}\ A$ and $\mathsf{is\_tm}\ M$, whose formation rules are as follows:

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

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

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

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

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

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.

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 the 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 on page 1510 using explicit contexts in Section 2.4. As another example of using explicit contexts, we give the standard typing rules for the polymorphic lambda-calculus (see Section 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 (Girard *et al.* 1990; Pierce 2002). 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.

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

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.

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

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

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

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 is in contrast to viewing contexts as multi-sets.

We thus introduce the following categories:

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

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:

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

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 $S_{\alpha t}$. Although a schema does not appear to have an explicit binder, all the eigenvariables and schematic variables occurring are considered bound. 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 (Cave and Pientka 2012; Pientka and Dunfield 2008).

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 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 has_schema S_l, where subscript l is used to denote the relationship between the schema and an instance of it.

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

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

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 $\mathsf{EV}(D) \cap \mathsf{EV}(\Gamma) = \varnothing$ 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 for example the inference rules in Section 2.4. We remark that a given context can in principle inhabit different schemas; for example, the context is_tp α_1, is_tp α_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 Section 4. 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 Section 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,

[†] Existing metalanguages make similar commitments to structural properties: for example, the LF-type theory satisfies by construction those properties and so does a specification logic based on hereditary Harrop formulae, as we elaborate in the companion paper (Felty *et al.* 2015a)

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 2.1 (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 2.1 (Structural properties of declarations).

1. Declaration weakening
$$\frac{\Gamma, \mathsf{rm}_A(D), \Gamma' \vdash J}{\Gamma, D, \Gamma' \vdash J}\ d\text{-wk}\,.$$
2. Declaration strengthening
$$\frac{\Gamma, D, \Gamma' \vdash J}{\Gamma, \mathsf{rm}_A(D), \Gamma' \vdash J}\ d - \mathrm{str}\dagger$$
with the proviso (†) that A is irrelevant to J and Γ'. In practice, this may be done by maintaining a dependency call graph of all judgments.
3. Declaration exchange
$$\frac{\Gamma, D, \Gamma' \vdash J}{\Gamma, \mathsf{perm}_\pi(D), \Gamma' \vdash J}\ d\text{-exc}\,.$$

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

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

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 2.2 (Exchange).

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

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

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 2.3 (Structural properties of contexts).

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

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.

— $\mathsf{rm}^{*}_{\mathsf{is\_tm}\ y; y:A}(\mathsf{is\_tp}\ \alpha,\ \mathsf{is\_tp}\ \beta) = (\mathsf{is\_tp}\ \alpha,\ \mathsf{is\_tp}\ \beta)$. An 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 2.4 (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 $\Gamma_1, (D_1; \mathit{is\_wf}\ x; D_2), \Gamma_2 \vdash J$, then $\Gamma_1, (D_1; [t/x]D_2), [t/x]\Gamma_2 \vdash [t/x]J$ for any term t for which $\Gamma_1, D_1 \vdash \mathit{is\_wf}\ t$ 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 substructural 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 on page 1510. The need for explicit structured contexts, as discussed in Sections 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 Section 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.

In this formulation, depicted in Figure 1 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,

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

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

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

Fig. 1. Explicit-context formulation of inference rules.

$x{:}B \in \Gamma$ 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 it is 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. $\Phi_{\alpha x} := \mathsf{rm}^*_{x:C}(\Phi_{\alpha t})$, $\Phi_{\alpha x}\ \mathsf{has\_schema}\ S_{\alpha x}$, and $\Phi_{\alpha x} \vdash \mathsf{is\_tm}\ M$;
2. $\Phi_\alpha := \mathsf{rm}^*_{\mathsf{is\_tm}\ x;x:C}(\Phi_{\alpha t})$, $\Phi_\alpha\ \mathsf{has\_schema}\ S_\alpha$, 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. $\Phi_{\alpha x} \sim \Phi_{\alpha t}$, $\Phi_{\alpha x}$ has_schema $S_{\alpha x}$, and $\Phi_{\alpha x} \vdash \mathsf{is\_tm}\ M$;
2. $\Phi_{\alpha} \sim \Phi_{\alpha t}$, Φ_{α} has_schema S_{α}, 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 context. 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 Section 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 the following.

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. As we go along, we 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 (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

$$\frac{\mathsf{aeq}\ x\ x \in \Gamma}{\Gamma \vdash \mathsf{aeq}\ x\ x}\ \mathrm{ae}_v \qquad \frac{\Gamma, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M\ N}{\Gamma \vdash \mathsf{aeq}\ (\mathsf{lam}\ x.\ M)\ (\mathsf{lam}\ x.\ N)}\ \mathrm{ae}_l\ .$$

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

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

$$\text{Context schemas} \quad \begin{array}{lcl} S_x & := & \mathsf{is\_tm}\ x \\ S_{xa} & := & \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \end{array} .$$

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 Section 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 Section 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

$$\frac{}{\cdot \sim \cdot}\ \mathrm{crel}_e \qquad \frac{\Phi_x \sim \Phi_{xa}}{\Phi_x, \mathsf{is\_tm}\ x \sim \Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x}\ \mathrm{crel}_{xa} .$$

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 3.1 (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 3.1 (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

$$\mathcal{D} = \frac{\mathsf{is\_tm}\ x \in \Phi_x}{\Phi_x \vdash \mathsf{is\_tm}\ x}\ \mathrm{tm}_v$$

$\mathsf{is\_tm}\ x \in \Phi_x$	by rule premise
$\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \in \Phi_{xa}$	by Lemma 3.1
$\Phi_{xa} \vdash \mathsf{aeq}\ x\ x$	by rule ae_v

Case

$$\mathcal{D} = \frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Phi_x \vdash \mathsf{is\_tm}\ M_1 & \Phi_x \vdash \mathsf{is\_tm}\ M_2\end{array}}{\Phi_x \vdash \mathsf{is\_tm}\ (\mathsf{app}\ M_1\ M_2)}\ \mathrm{tm}_a$$

$\Phi_x \vdash \mathsf{is\_tm}\ M_1$	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

$$\mathcal{D} = \frac{\mathcal{D}'}{\Phi_x \vdash \mathsf{is\_tm}\ (\mathsf{lam}\, x.\, M)}\ \mathrm{tm}_l$$

$\Phi_x, \mathsf{is\_tm}\ x \vdash \mathsf{is\_tm}\ M$	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.

To be precise about the instantiation of the inductions hypothesis, consider the following general statement of the theorem (and induction hypothesis):

$$\text{forall } \Phi_1, \Phi_2, N, \text{ if } \Phi_1 \sim \Phi_2 \text{ and } \Phi_1 \vdash \mathsf{is\_tm}\ N, \text{ then } \Phi_2 \vdash \mathsf{aeq}\ N\ N.$$

In the tm_l case above, Φ_1, Φ_2 and N in the conclusion of this case are Φ_x, Φ_{xa} and $(\mathsf{lam}\, x.\, M)$, respectively, while the instantiations of Φ_1, Φ_2 and N for the induction hypothesis are $(\Phi_x, \mathsf{is\_tm}\ x)$, $(\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x)$ and M, respectively.

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 3.2 (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

$$\mathcal{D} = \frac{\mathsf{is\_tm}\ x \in \Phi_{xa}}{\Phi_{xa} \vdash \mathsf{is\_tm}\ x}\ \mathrm{tm}_v$$

$\mathsf{is\_tm}\ x \in \Phi_{xa}$ 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

$$\mathcal{D} = \frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Phi_{xa} \vdash \mathsf{is\_tm}\ M_1 & \Phi_{xa} \vdash \mathsf{is\_tm}\ M_2\end{array}}{\Phi_{xa} \vdash \mathsf{is\_tm}\ (\mathsf{app}\ M_1\ M_2)}\ \mathrm{tm}_a$$

$\Phi_{xa} \vdash \mathsf{aeq}\ M_1\ M_1$ 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

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

$\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{is\_tm}\ M$ 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

We again consider the general statement of the theorem (and induction hypothesis):

$$\text{forall } \Phi, N,\ \text{if } \Phi \vdash \mathsf{is\_tm}\ N,\ \text{then } \Phi \vdash \mathsf{aeq}\ N\ N.$$

In this version of the tm_l case, Φ and N in the conclusion are Φ_{xa} and $(\mathsf{lam}\, x.\, M)$, respectively, while the instantiations of Φ and N for the induction hypothesis are $(\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x)$ and M, respectively.

Note that the application cases of Theorems 3.1 and 3.2 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 Section 3.3. Since the proof involves only Φ_{xa}, the two approaches (R & G) collapse.

Lemma 3.2 (Context inversion). If $\mathsf{aeq}\ M\ N \in \Phi_{xa}$, then $M = N$.

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

Theorem 3.3 (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 3.2 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 Section 2, the following new context schemas are also used here

$$\begin{array}{lcl} S_{\alpha p} & := & \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha \\ S_{\alpha\alpha q} & := & \mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha\ +\ \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \end{array} .$$

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

Algorithmic equality for the polymorphic lambda-calculus

$$\cdots$$

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

$$\frac{\Gamma \vdash \mathsf{aeq}\ M\ N \quad \Gamma \vdash \mathsf{atp}\ A\ B}{\Gamma \vdash \mathsf{aeq}\ (\mathsf{tapp}\ M\ A)\ (\mathsf{tapp}\ N\ B)}\ \mathrm{ae}_{ta} \quad .$$

$$\frac{\mathsf{atp}\ \alpha\ \alpha \in \Gamma}{\Gamma \vdash \mathsf{atp}\ \alpha\ \alpha}\ \mathrm{at}_{\alpha}$$

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

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 $S_{\alpha p}$ since the statement and proof do not depend on terms. The schema for the latter is $S_{\alpha\alpha q}$.

Theorem 3.4 (Admissibility of reflexivity for types, G version). If $\Phi_{\alpha p} \vdash \mathsf{is\_tp}\ A$, then $\Phi_{\alpha p} \vdash \mathsf{atp}\ A\ A$.

The proof is exactly the same as the proof of Theorem 3.2, modulo replacing app and lam with arr and all, respectively, and using the corresponding rules.

As we have already mentioned in Section 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 3.3 (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 3.4
$\Phi_{aeq} \vdash \mathsf{atp}\ A\ A$	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 3.2 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 3.4 and instead prove Lemma 3.3 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 Section 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 3.5 (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 3.2. We show the case for application of terms to types.

Case

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

$\Phi_{aeq} \vdash \mathsf{aeq}\ M\ M$	by IH on $\mathcal{D}_1$
$\Phi_{aeq} \vdash \mathsf{atp}\ A\ A$	by Lemma 3.3 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 $\Phi_\alpha \sim \Phi_{\alpha p}$, $\Phi_{\alpha x} \sim \Phi_{\alpha q}$, $\Phi_{\alpha x} \sim \Phi_\alpha$ and $\Phi_{\alpha q} \sim \Phi_{\alpha p}$. We define the first two as follows (where we omit the inference rules for the base cases).

Context relations

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

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

Note that $\Phi_{\alpha x} \sim \Phi_{\alpha q}$ is the extension of $\Phi_x \sim \Phi_{xa}$ with one additional case for equality for types. Again, we remark on our policy to use the smallest contexts possible for modularity reasons. Otherwise, we could have omitted the $\Phi_\alpha \sim \Phi_{\alpha p}$ relation, and stated the next theorem using $\Phi_{\alpha x} \sim \Phi_{\alpha q}$. We also omit the (obvious) inference rules defining $\Phi_{\alpha x} \sim \Phi_\alpha$ and $\Phi_{\alpha q} \sim \Phi_{\alpha p}$, 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 3.1) and is therefore omitted.

Theorem 3.6 (Admissibility of reflexivity for types, R version).
Let $\Phi_\alpha \sim \Phi_{\alpha p}$. If $\Phi_\alpha \vdash \mathsf{is\_tp}\ A$, then $\Phi_{\alpha p} \vdash \mathsf{atp}\ A\ A$.

Lemma 3.4 (Relational strengthening). Let $\Phi_{\alpha x} \sim \Phi_{\alpha q}$. Then, there exist contexts Φ_α and $\Phi_{\alpha p}$ such that $\Phi_{\alpha x} \sim \Phi_\alpha$, $\Phi_{\alpha q} \sim \Phi_{\alpha p}$ and $\Phi_\alpha \sim \Phi_{\alpha p}$.

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

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

Lemma 3.5 (R-promotion for type reflexivity). Let $\Phi_{\alpha x} \sim \Phi_{\alpha q}$. If $\Phi_{\alpha x} \vdash \mathsf{is\_tp}\ A$, then $\Phi_{\alpha q} \vdash \mathsf{atp}\ A\ A$.

Proof.

$\Phi_{\alpha x} \vdash \mathsf{is\_tp}\ A$	by assumption
$\Phi_\alpha \vdash \mathsf{is\_tp}\ A$	by c-str
$\Phi_{\alpha x} \sim \Phi_{\alpha q}$	by assumption
$\Phi_\alpha \sim \Phi_{\alpha p}$	by relational strengthening (Lemma 3.4)
$\Phi_{\alpha p} \vdash \mathsf{atp}\ A\ A$	by Theorem 3.6
$\Phi_{\alpha q} \vdash \mathsf{atp}\ A\ A$	by c-wk

Theorem 3.7 (Admissibility of reflexivity for terms, R version). Let $\Phi_{\alpha x} \sim \Phi_{\alpha\alpha q}$. If $\Phi_{\alpha x} \vdash \mathsf{is\_tm}\ M$, then $\Phi_{\alpha\alpha q} \vdash \mathsf{aeq}\ M\ M$.

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 3.1) and the G version for types in the polymorphic case (Theorem 3.4). We show again the case for application of terms to types to compare with the G version.
Case

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

$\Phi_{\alpha x} \sim \Phi_{\alpha\alpha q}$	by assumption
$\Phi_{\alpha x} \vdash \mathsf{is\_tm}\ M$	sub-derivation $\mathcal{D}_1$
$\Phi_{\alpha\alpha q} \vdash \mathsf{aeq}\ M\ M$	by IH
$\Phi_{\alpha x} \vdash \mathsf{is\_tp}\ A$	sub-derivation $\mathcal{D}_2$
$\Phi_{\alpha\alpha q} \vdash \mathsf{atp}\ A\ A$	by Lemma 3.5
$\Phi_{\alpha\alpha q} \vdash \mathsf{aeq}\ (\mathsf{tapp}\ M\ A)\ (\mathsf{tapp}\ M\ A)$	by rule ae_{ta}

Since type equality is subordinate to term equality we can pursue reflexivity of the former independently. The context relation $\Phi_{tp} \sim \Phi_{\alpha p}$ is defined analogously to $\Phi_x \sim \Phi_{xa}$ and so we omit it.

Lemma 3.6 ('Member' lemma for type equality). If $\Phi_{tp} \sim \Phi_{\alpha p}$, then $\mathsf{is\_tp}\ \alpha \in \Phi_{tp}$ iff $\mathsf{atp}\ \alpha\ \alpha \in \Phi_{\alpha p}$.

Proof. Standard. □

Theorem 3.8 (Admissibility of reflexivity for types, R version). Let $\Phi_{tp} \sim \Phi_{\alpha p}$. If $\Phi_{tp} \vdash \mathsf{is\_tp}\ A$, then $\Phi_{\alpha p} \vdash \mathsf{atp}\ A\ A$.

Proof. By induction on the given derivation. □

Now the context relation $\Phi_x \sim \Phi_{xa}$, relative to the proof of admissibility of reflexivity, has an additional case.

Context relation

$$\frac{}{\cdot \sim \cdot}\ \mathit{crel}_e \qquad \frac{\Phi_x \sim \Phi_{\alpha\alpha q}}{\Phi_x, \mathsf{is\_tm}\ x \sim \Phi_{\alpha\alpha q}, \mathsf{aeq}\ x\ x}\ \mathrm{crel}_{tm} \qquad \frac{\Phi_x \sim \Phi_{\alpha\alpha q}}{\Phi_x, \mathsf{is\_tp}\ a \sim \Phi_{\alpha\alpha q}, \mathsf{atp}\ \alpha\ \alpha}\ \mathrm{crel}_{tp}$$

Lemma 3.7 ('Member' lemma for term equality). Let $\Phi_x \sim \Phi_{\alpha\alpha q}$. $\mathsf{is\_tm}\ x \in \Phi_x$ iff $\mathsf{aeq}\ x\ x \in \Phi_{\alpha\alpha q}$.

Proof. Standard. □

Theorem 3.9 (Admissibility of reflexivity for terms, R version). Let $\Phi_x \sim \Phi_{\alpha\alpha q}$. If $\Phi_x \vdash \mathsf{is\_tm}\ M$, then $\Phi_{\alpha\alpha q} \vdash \mathsf{aeq}\ M\ M$.

Proof. By induction on the given derivation using the above Theorem 3.8 in the ae_{ta} case. □

3.3. *Non-linear context extensions*

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

Declarative equality

$$\frac{\mathsf{deq}\ x\ x \in \Gamma}{\Gamma \vdash \mathsf{deq}\ x\ x}\ \mathrm{de}_v \qquad \frac{\Gamma, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x \vdash \mathsf{deq}\ M\ N}{\Gamma \vdash \mathsf{deq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)}\ \mathrm{de}_l$$

$$\frac{\Gamma \vdash \mathsf{deq}\ M_1\ N_1 \quad \Gamma \vdash \mathsf{deq}\ M_2\ N_2}{\Gamma \vdash \mathsf{deq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}\ \mathrm{de}_a \quad .$$

$$\frac{}{\Gamma \vdash \mathsf{deq}\ M\ M}\ \mathrm{de}_r \qquad \frac{\Gamma \vdash \mathsf{deq}\ M\ L \quad \Gamma \vdash \mathsf{deq}\ L\ N}{\Gamma \vdash \mathsf{deq}\ M\ N}\ \mathrm{de}_t \qquad \frac{\Gamma \vdash \mathsf{deq}\ N\ M}{\Gamma \vdash \mathsf{deq}\ M\ N}\ \mathrm{de}_s$$

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:

$$\text{Generalized context schema} \quad S_{da} := \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x; \mathsf{aeq}\ x\ x$$

The following lemma promotes Theorems 3.2 and 3.3 to the 'bigger' generalized context.

Lemma 3.8 (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 3.3 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 3.10 (Completeness, G version).
If $\Phi_{da} \vdash \mathsf{deq}\ M\ N$, then $\Phi_{da} \vdash \mathsf{aeq}\ M\ N$.

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

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

$$\mathcal{D} = \frac{}{\Phi_{da} \vdash \mathsf{deq}\ M\ M}\ \mathrm{de}_r$$

$\Phi_{da} \vdash \mathsf{is\_tm}\ M$ by (implicit) assumption
$\Phi_{da} \vdash \mathsf{aeq}\ M\ M$ by Lemma 3.8 (1)
Case

$$\mathcal{D} = \frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Phi_{da} \vdash \mathsf{deq}\ M\ L & \Phi_{da} \vdash \mathsf{deq}\ L\ N\end{array}}{\Phi_{da} \vdash \mathsf{deq}\ M\ N}\ \mathrm{de}_t$$

$\Phi_{da} \vdash \mathsf{aeq}\ M\ L$ 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 3.8 (3)

Case

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

$\Phi_{da}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{deq}\ M\ N$ 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 3.8 (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 3.2, 3.4 and 3.5) 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

$$\frac{\Phi_{xa} \sim \Phi_{xd}}{\Phi_{xa}, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \sim \Phi_{xd}, \mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x}\ \mathrm{crel}_{ad}\ .$$

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

Lemma 3.9 (Relational strengthening). Let $\Phi_{xa} \sim \Phi_{xd}$. Then, there exists a context Φ_x such that $\Phi_x \sim \Phi_{xa}$.

Lemma 3.10 (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 3.4 and 3.5, with the proof of Lemma 3.10 requiring Lemma 3.9.

Theorem 3.11 (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

$$\mathcal{D} = \frac{}{\Phi_{xd} \vdash \mathsf{deq}\ M\ M}\ \mathrm{de}_r$$

$\Phi_{xd} \vdash \mathsf{is\_tm}\ M$ by (implicit) assumption
$\Phi_{xa} \vdash \mathsf{aeq}\ M\ M$ by Theorem 3.10

Case

$$\mathcal{D} = \frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Phi_{xd} \vdash \mathsf{deq}\ M\ L & \Phi_{xd} \vdash \mathsf{deq}\ L\ N\end{array}}{\Phi_{xd} \vdash \mathsf{deq}\ M\ N}\ \mathrm{de}_t$$

$\Phi_{xa} \vdash \mathsf{aeq}\ M\ L$ 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 3.3 (2)

Case

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

$\Phi_{xa} \sim \Phi_{xd}$ 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 3.10) uses 3 occurrences of each of c-str and c-wk via promotion Lemma 3.8 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 3.11) does.

Theorem 3.12 (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

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

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

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

$\Phi_{xa}, \mathsf{is\_tm}\ y; \mathsf{aeq}\ y\ y, \mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x \vdash \mathsf{aeq}\ M_1\ M_2$ 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 3.12 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.

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

The typing rules are the obvious subset of the ones presented in Section 2, modified to take a type superscript on bound variables, yielding

$$\text{Context schema} \quad S_t := \mathsf{is\_tm}\ x; x{:}A \quad .$$

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. Recall that we assume that eigenvariables in different blocks in a context satisfying the schema are distinct from each other.

Theorem 3.13 (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 the reader 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

$$\frac{x \rightsquigarrow x \in \Gamma}{\Gamma \vdash x \rightsquigarrow x}\ \mathrm{pr}_v \qquad \frac{\Gamma, \mathsf{is\_tm}\ x; x \rightsquigarrow x \vdash M \rightsquigarrow N}{\Gamma \vdash \mathsf{lam}\, x.\, M \rightsquigarrow \mathsf{lam}\, x.\, N}\ \mathrm{pr}_l$$

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

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

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

The relevant substitution lemma is:

Lemma 3.11. 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 Section 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

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

Theorem 3.14 (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

$$\mathcal{D} = \frac{x \rightsquigarrow x \in \Phi_r}{\Phi_r \vdash x \rightsquigarrow x}\ \mathrm{pr}_v.$$

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

Case

$$\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Phi_r, \mathsf{is\_tm}\ x; x \rightsquigarrow x \vdash M \rightsquigarrow M' & \Phi_r \vdash N \rightsquigarrow N'\end{array}}{\Phi_r \vdash (\mathsf{app}\ (\mathsf{lam}\, x.\, M)\ N) \rightsquigarrow [N'/x]M'}\ \mathrm{pr}_\beta$$

$\Phi_t \vdash (\mathsf{app}\ (\mathsf{lam}\, x.\, M)\ N) : A$	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 3.11 (substitution)

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

Lemma 3.12.
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. Conclusions

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.

As mentioned, in a related paper (Felty *et al.* 2015a), we compare four systems on the benchmarks presented here. We refer the reader to this paper for the details of the formalizations in Twelf, Beluga, Hybrid and Abella, and for an extensive discussion of their comparison, along with a summary table comparing these systems on 13 features that we have identified. To give a flavour of this comparison, we mention two general points of comparison here. First, our results show that Beluga and Twelf are better suited to G versions of the theorems, whereas Abella and Hybrid are better suited to R versions. Furthermore, R versions are possible in Beluga, but not in Twelf, while G versions are possible to varying degrees in both Hybrid and Abella.

Another point of comparison is how much general support for HOAS (and beyond) that each system supports. In summary, Beluga provides intrinsic support for abstracting over variables and contexts, as well as for relating contexts via first-class substitutions and inductive definitions. Abella includes a special ∇-quantifier to abstract over objects denoting variables and also provides inductive definitions. Contexts in both Abella

and Hybrid must be handled explicitly. Hybrid lacks intrinsic support for abstracting over variables, which increases the burden on the user. While this is a significant drawback, Hybrid has several advantages. It inherits both inductive definitions and recursive functions from the ambient logic, which simplify proofs about context schemas and relations. In addition, it can directly take advantage of Coq's tactics, libraries and decision procedures.

This comparison has had the additional benefit of leading to or validating planned improvements to the systems. For example, Beluga now has better support for context relations, and there is continued work in this direction. One of the next steps for Hybrid is to implement the ∇ quantifier, which will provide greatly increased support for context variables. There has also been new work in Abella in the direction of providing better support for automating lemmas about contexts.

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.

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 endeavour. A more recent version can be found in Claessen *et al.* (2015). On the other hand, Twelf's wiki (`http://twelf.org/wiki/Case_studies`), Abella's library (`http://abella-prover.org/examples`), Beluga's distribution and the Coq implementation of Hybrid (`http://www.site.uottawa.ca/~afelty/HybridCoq/`) contain a set of context-intensive examples, some of which coincide with the ones presented here.

Future Work. 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 Section 3. We agree and we mention some additional categories, which we could not discuss in the present paper for the sake of space.

— 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. Thus, systems like Hybrid could take advantage of such procedures. At the same time, 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. Case in point, the proof of correctness for the translation between De Bruijn and HOAS terms in Abella, see `http://abella-prover.org/examples/lambda-calculus/debruijn_ho.thm`, 40% of which consists of basic facts about natural numbers.
— 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) or other case studies such as Wang *et al.* (2013), where contexts have more structure, as they are induced by *third-order* specifications.
— 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 encoding 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 (2015) and Gacek *et al.* (2012).
— Finally, a subject that is gaining importance is the encoding of *infinite* behaviour, typically realized via some form of *co-induction*. Context-intensive case studies have been explored, for example, in Momigliano (2012); Momigliano *et al.* (2002).

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 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. Another direction is *abstracting* over the structure of contexts, which is now tied up to sequences, possibly in the form of an abstract data-type of context construction that satisfies certain properties w.r.t. the rm and perm operations. This could help to capture more exotic context structures, such as the ones occurring in the logic of bunched implication.

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.

Appendix. 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	3.1	R	1522
aeq-reflexivity for ULC	3.2	G	1523
aeq-symmetry and transitivity for ULC	3.3	same	1524
atp-reflexivity for POLY	3.4	G	1524
aeq-reflexivity for POLY	3.5	G	1525
atp-reflexivity for POLY	3.8	R	1526
aeq-reflexivity for POLY	3.9	R	1527
aeq/deq-completeness for ULC	3.10	G	1528
aeq/deq-completeness for ULC	3.11	R	1530
type uniqueness for STLC	3.13	same	1532
type preservation for parallel reduction for STLC	3.14	R	1533
aeq-parallel substitution for ULC	3.12	same	1531

A.2. A recap of schemas and their usage.

Context	Schema	Block	Description/Used in:
Φ_{α}	S_{α}	is_tp α	type variables
Φ_{x}	S_{x}	is_tm x	term variables
$\Phi_{\alpha x}$	$S_{\alpha x}$	is_tp α + is_tm x	type/term variables
$\Phi_{\alpha t}$	$S_{\alpha t}$	is_tp α + is_tm x; $x{:}T$	type-checking for POLY
Φ_{xa}	S_{xa}	is_tm x; aeq x x	Thm 3.2, 3.3, and 3.12
$\Phi_{\alpha tp}$	$S_{\alpha tp}$	is_tp α; atp α α	Thm 3.4
$\Phi_{\alpha aq}$	$S_{\alpha aq}$	is_tp α; atp α α + is_tm x; aeq x x	Thm 3.5
Γ_{da}	S_{da}	is_tm x; deq x x; aeq x x	Thm 3.10
Φ_{xd}	S_{xd}	is_tm x; deq x x	Thm 3.11
Φ_{t}	S_{t}	is_tm x; oft x A	Thm 3.13
Φ_{r}	S_{r}	is_tm x; $x \rightsquigarrow x$	Thm 3.14

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 3.1
$\Phi_\alpha \sim \Phi_{\alpha p}$	$\mathsf{is\_tp}\ \alpha \sim (\mathsf{is\_tp}\ \alpha; \mathsf{atp}\ \alpha\ \alpha)$	Thm 3.6
$\Phi_{\alpha x} \sim \Phi_{\alpha eq}$	$\Phi_x \sim \Phi_{xa}$ plus $\Phi_\alpha \sim \Phi_{\alpha p}$	Thm 3.7
$\Phi_{xa} \sim \Phi_{xd}$	$(\mathsf{is\_tm}\ x; \mathsf{aeq}\ x\ x) \sim (\mathsf{is\_tm}\ x; \mathsf{deq}\ x\ x)$	Thm 3.11
$\Phi_r \sim \Phi_t$	$(\mathsf{is\_tm}\ x; x \rightsquigarrow x) \sim (\mathsf{is\_tm}\ x; x{:}A)$	Thm 3.14

References

Aydemir, B.E., Bohannon, A., Fairbairn, M., Foster, J.N., Pierce, B.C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S. and Zdancewic, S. (2005). Mechanized metatheory for the masses: The POPLMARK challenge. In: *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, vol. 3603, Springer, 50–65.

Cave, A. and Pientka, B. (2012). Programming with binders and indexed data-types. In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM Press, 413–424.

Cave, A. and Pientka, B. (2015). A case study on logical relations using contextual types. In: *Proceedings of the 10th International Workshop on Logical Frameworks and Meta Languages: Theory and Practice*, *LFMTP 2015*, Electronic Proceedings in Theoretical Computer Science, vol. 185, 33–45.

Claessen, K., Johansson, M., Rosén, D. and Smallbone, N. (2015). TIP: Tons of inductive problems. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F. and Sorge, V. (eds.) *Proceedings of the Intelligent Computer Mathematics - International Conference, CICM 2015*, Washington, DC, USA, July 13–17, 2015, Lecture Notes in Computer Science, vol. 9150, Springer, 333–337.

Crary, K. (2009). Explicit contexts in LF (extended abstract). In: *Proceedings of the 3rd International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP 2008*, Electronic Proceedings in Theoretical Computer Science, vol. 228, Elsevier, 53–68.

Felty, A., Momigliano, A. and Pientka, B. (2015a). The next 700 challenge problems for reasoning with higher-order abstract syntax representations: Part 2—a survey. *Journal of Automated Reasoning* **55** (4) 307–372.

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

Felty, A.P., Momigliano, A. and Pientka, B. (2015b). An open challenge problem repository for systems supporting binders. In: *Proceedings of the 10th International Workshop on Logical Frameworks and Meta Languages: Theory and Practice, LFMTP 2015*, Electronic Proceedings in Theoretical Computer Science, vol. 185, pages 18–32.

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

Gacek, A. (2008). The Abella interactive theorem prover (system description). In: *Proceedings of the 4th International Joint Conference on Automated Reasoning*, Lecture Notes in Computer Science, vol. 5195, 154–161.

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

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

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

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

Hoos, H.H. and Stützle, T. (2000). Satlib: An online resource for research on SAT. In: Gent, I., Maaren, H.V. and Walsh, T. (eds.) *SAT 2000: Highlights of Satisfiability Research in the Year 2000*, Frontiers in Artificial Intelligence and Applications, vol. 63, IOS Press, 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. and 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*, Lecture Notes in Computer Science, vol. 1862, Springer, 411–426.

Momigliano, A. (2012). A supposedly fun thing I may have to do again: A HOAS encoding of Howe's method. In: *Proceedings of the 7th ACM SIGPLAN International Workshop on Logical Frameworks and Meta-Languages, Theory and Practice*, ACM Press, 33–42.

Momigliano, A., Ambler, S. and Crole, R.L. (2002). A Hybrid encoding of Howe's method for establishing congruence of bisimilarity. *Electronic Notes in Theoretical Computer Science* **70** (2) 60–75.

Momigliano, A., Martin, A.J. and Felty, A.P. (2008). Two-level Hybrid: A system for reasoning using higher-order abstract syntax. In: *Proceedings of the 2nd International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP 2008*, Electronic Notes in Theoretical Computer Science, vol. 196, Elsevier, 85–93.

Pfenning, F. (2001). Computation and deduction. `http://www.cs.cmu.edu/~fp/courses/comp-ded/handouts/cd.pdf`, Accessed 26 October 2016.

Pientka, B. (2007). Proof pearl: The power of higher-order encodings in the logical framework LF. In: *Proceedings of the 20th International Conference on Theorem Proving in Higher-Order Logics*, Lecture Notes in Computer Science, Springer, 246–261.

Pientka, B. (2008). A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In: *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages*, ACM Press, 371–382.

Pientka, B. and Cave, A. (2015). Inductive Beluga: Programming proofs (system description). In: Felty, A.P. and Middeldorp, A. (eds.) *Proceedings of the 25th International Conference on Automated Deduction (CADE-25)*, Lecture Notes in Computer Science, vol. 9195, Springer, 272–281.

Pientka, B. and Dunfield, J. (2008). Programming with proofs and explicit contexts. In: *Proceedings of the 10th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, ACM Press, 163–173.

Pientka, B. and Dunfield, J. (2010). Beluga: A framework for programming and reasoning with deductive systems (system description). In: *Proceedings of the 5th International Joint Conference on Automated Reasoning*, Lecture Notes in Computer Science, vol. 6173, Springer, 15–21.

Pierce, B.C. (2002). *Types and Programming Languages*, MIT Press.

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

Poswolsky, A. B. and Schürmann, C. (2008). Practical programming with higher-order encodings and dependent types. In: *Proceedings of the 17th European Symposium on Programming*, Lecture Notes in Computer Science, vol. 4960, Springer, 93–107.

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: *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, vol. 5674, Springer, 79–83.

Schürmann, C. and Pfenning, F. (2003). A coverage checking algorithm for LF. In: *Prceedings 16th International Conference on Theorem Proving in Higher Order Logics*, Lecture Notes in Computer Science, vol. 2758, Springer, 120–135.

Schürmann, C. and Sarnat, J. (2008). Structural logical relations. In: *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society, 69–80.

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

Wang, Y., Chaudhuri, K., Gacek, A. and Nadathur, G. (2013). Reasoning about higher-order relational specifications. In: *Proceedings of the 15th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, ACM Press, 157–168.

Wang, Y. and Nadathur, G. (2013). Towards extracting explicit proofs from totality checking in Twelf. In: *Proceedings of the 8th ACM SIGPLAN International Workshop on Logical Frameworks and Meta-languages: Theory and Practice*, ACM Press, 55–66.