Reasoning with Higher-Order Abstract Syntax and Contexts: A Comparison

Amy Felty

University of Ottawa

July 13, 2010

Joint work with Brigitte Pientka, McGill University

Comparing Systems

- We focus on logical frameworks that support the use of *higher-order abstract syntax*.
  - Commonalities:
    - ★ encode object-level binders with meta-level binders
    - ★ support for alpha-renaming and substitution
    - ★ encode axioms and inference rules using *hypothetical and parametric judgments*
  - Differences:
    - ★ how a system supports reasoning *about* hypothetical and parametric derivations, which requires support for *contexts* to keep track of hypotheses
    - ★ other features...
- Systems studied here include Twelf, Beluga, and Hybrid.

Case Studies

- In the domain of meta-theory of programming languages
- Designed to highlight the differences and help practitioners in choosing a system
- Purposely simple, so they can be easily understood, and one can quickly appreciate the capabilities and trade-offs of different systems

Criteria for Comparison

- How do we represent contexts in proofs?
- How do we reason with contexts?
- How do we retrieve elements from a context?
- How easy is it to state a given theorem?
- How do we apply a substitution lemma?
- How do we know we have implemented a proof?
- How easy is it to interface the system with, for example, support for natural numbers?

Outline

- Example: Equivalence of Algorithmic and Declarative Equality
  - Representing Syntax of the Untyped Lambda Calculus
  - Encoding Inference Rules (in Twelf and Beluga)
  - Beluga Proof
  - Hybrid Proof
- Comparison
- Other Benchmarks and Conclusion

Representing Syntax

Object Logic

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

Example: $\mathsf{lam}\, x.\, \mathsf{lam}\, y.\, \mathsf{app}\ x\ y$

Twelf and Beluga

```
exp: type
lam: (exp -> exp) -> exp
app: exp -> exp -> exp
```

Example: `lam (λx.lam (λy.app x y))`

Representing Syntax in Hybrid

- Hybrid is implemented in Coq and Isabelle/HOL.
- Provides an inductively defined type *expr* and some operators so that object-level terms can be encoded in the same style as Twelf (except that they are "untyped").
- The user works with the higher-order syntax:
  $\mathsf{lam}\, x.\, \mathsf{lam}\, y.\, \mathsf{app}\ x\ y$.
- Unfolding definitions reveals an underlying de Bruijn representation.
- This representation is built definitionally on the foundation of the meta-language of the underlying theorem prover; no axioms are introduced.

Algorithmic and Declarative Equality

Context $\Psi ::= \cdot \mid \Psi, \mathsf{eq}\ \mathtt{x}\ \mathtt{x}$

$$\frac{\mathsf{eq}\ x\ x \in \Psi}{\Psi \vdash \mathsf{eq}\ x\ x} \qquad \frac{\Psi, \mathsf{eq}\ x\ x \vdash \mathsf{eq}\ M\ N}{\Psi \vdash \mathsf{eq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)}$$

$$\frac{\Psi \vdash \mathsf{eq}\ M_1\ N_1 \quad \Psi \vdash \mathsf{eq}\ M_2\ N_2}{\Psi \vdash \mathsf{eq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}$$

Algorithmic and Declarative Equality

Context $\Psi ::= \cdot \mid \Psi, \mathtt{eq\ x\ x}$

$$\frac{\mathsf{eq}\ x\ x \in \Psi}{\Psi \vdash \mathsf{eq}\ x\ x} \qquad \frac{\Psi, \mathsf{eq}\ x\ x \vdash \mathsf{eq}\ M\ N}{\Psi \vdash \mathsf{eq}\ (\mathsf{lam}\, x.\ M)\ (\mathsf{lam}\, x.\ N)}$$

$$\frac{\Psi \vdash \mathsf{eq}\ M_1\ N_1 \quad \Psi \vdash \mathsf{eq}\ M_2\ N_2}{\Psi \vdash \mathsf{eq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}$$

Context $\Phi ::= \cdot \mid \Phi, \mathtt{equal\ x\ x}$

$$\frac{\mathsf{equal}\ x\ x \in \Phi}{\Phi \vdash \mathsf{equal}\ x\ x} \qquad \frac{\Phi, \mathsf{equal}\ x\ x \vdash \mathsf{equal}\ M\ N}{\Phi \vdash \mathsf{equal}\ (\mathsf{lam}\, x.\ M)\ (\mathsf{lam}\, x.\ N)}$$

$$\frac{\Phi \vdash \mathsf{equal}\ M_1\ N_1 \quad \Phi \vdash \mathsf{equal}\ M_2\ N_2}{\Phi \vdash \mathsf{equal}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}$$

$$\frac{}{\Phi \vdash \mathsf{equal}\ M\ M} \qquad \frac{\Phi \vdash \mathsf{equal}\ M\ L \quad \Phi \vdash \mathsf{equal}\ L\ N}{\Phi \vdash \mathsf{equal}\ M\ N}$$

Theorem 2 (Completeness)

- Attempt 1: If $\Phi \vdash \text{equal } M\ N$ then $\Psi \vdash \text{eq } M\ N$.
- Problem: this statement does not contain enough information about how the two contexts Φ and Ψ are related.

Theorem 2 (Completeness)

- Attempt 1: If $\Phi \vdash \text{equal } M\ N$ then $\Psi \vdash \text{eq } M\ N$.
- Problem: this statement does not contain enough information about how the two contexts Φ and Ψ are related.
- Solution 1: Add more info (later)
- Solution 2: Generalized context $\Gamma ::= \cdot \mid \Gamma, \text{eq } x\ x, \text{equal } x\ x$

Theorem 2 (Completeness)

- Attempt 1: If $\Phi \vdash \text{equal } M\ N$ then $\Psi \vdash \text{eq } M\ N$.
- Problem: this statement does not contain enough information about how the two contexts Φ and Ψ are related.
- Solution 1: Add more info (later)
- Solution 2: Generalized context $\Gamma ::= \cdot \mid \Gamma, \text{eq } x\ x, \text{equal } x\ x$
- Attempt 2: If $\Gamma \vdash \text{equal } M\ N$ then $\Gamma \vdash \text{eq } M\ N$.
- Proof of lambda case:

$\Gamma \vdash \text{equal } (\text{lam } x.\ M)\ (\text{lam } x.\ N)$	by assumption
$\Gamma, \text{equal } x\ x \vdash \text{equal } M\ N$	by decl. equality rule for lam
$\Gamma, \text{eq } x\ x, \text{equal } x\ x \vdash \text{equal } M\ N$	by weakening
$\Gamma, \text{eq } x\ x, \text{equal } x\ x \vdash \text{eq } M\ N$	by i.h.
$\Gamma, \text{eq } x\ x \vdash \text{eq } M\ N$	by strengthening
$\Gamma \vdash \text{eq } (\text{lam } x.\ M)\ (\text{lam } x.\ N)$	by alg. equality rule for lam

Inference Rules of the Object Logic (Again)

$$\frac{\Psi, \mathsf{eq}\ x\ x \vdash \mathsf{eq}\ M\ N}{\Psi \vdash \mathsf{eq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)} \qquad \frac{\Psi \vdash \mathsf{eq}\ M_1\ N_1 \quad \Psi \vdash \mathsf{eq}\ M_2\ N_2}{\Psi \vdash \mathsf{eq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}$$

Twelf and Beluga Encoding

$\mathtt{eq} : \mathtt{exp} \to \mathtt{exp} \to \mathtt{type}.$

$\mathtt{eq\_lam} : (\Pi \mathtt{x} : \mathtt{exp}.\ \mathtt{eq}\ \mathtt{x}\ \mathtt{x} \to \mathtt{eq}\ (\mathtt{E}\ \mathtt{x})\ (\mathtt{F}\ \mathtt{x}))$
$\qquad \to \mathtt{eq}\ (\mathtt{lam}\ (\lambda x.\ \mathtt{E}\ \mathtt{x}))\ (\mathtt{lam}\ (\lambda x.\ \mathtt{F}\ \mathtt{x})).$

$\mathtt{eq\_app} : \mathtt{eq}\ \mathtt{E1}\ \mathtt{F1} \to \mathtt{eq}\ \mathtt{E2}\ \mathtt{F2} \to \mathtt{eq}\ (\mathtt{app}\ \mathtt{E1}\ \mathtt{E2})\ (\mathtt{app}\ \mathtt{F1}\ \mathtt{F2}).$

Inference Rules of the Object Logic (Again)

$$\frac{\Psi, \mathsf{eq}\ x\ x \vdash \mathsf{eq}\ M\ N}{\Psi \vdash \mathsf{eq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)} \qquad \frac{\Psi \vdash \mathsf{eq}\ M_1\ N_1 \quad \Psi \vdash \mathsf{eq}\ M_2\ N_2}{\Psi \vdash \mathsf{eq}\ (\mathsf{app}\ M_1\ M_2)\ (\mathsf{app}\ N_1\ N_2)}$$

Twelf and Beluga Encoding

```
eq : exp → exp → type.
eq_lam : (Πx : exp. eq x x → eq (E x) (F x))
          → eq (lam (λx. E x)) (lam (λx. F x)).
eq_app : eq E1 F1 → eq E2 F2 → eq (app E1 E2) (app F1 F2).

equal : exp → exp → type.
e_l : (Πx : exp. equal x x → equal (E x) (F x))
          → equal (lam (λx. E x)) (lam (λx. F x)).
e_a : equal E1 F1 → equal E2 F2
          → equal (app E1 E2) (app F1 F2).
e_r : equal E E.
e_t : equal E E′ → equal E′ F → equal E F.
```

Beluga Proof of Theorem 2

- Context schemas classify contexts:

  $\text{schema}\ \texttt{eCtx} = \texttt{block}\ \texttt{x} : \texttt{exp},\ \texttt{u} : \texttt{eq}\ \texttt{x}\ \texttt{x}.\ \texttt{equal}\ \texttt{x}\ \texttt{x}$

- Inductive proofs about derivations are written as recursive functions using pattern matching.

$\text{rec}\ \texttt{ceq} : \{\gamma : \texttt{eCtx}\}(\texttt{equal}\ (\texttt{T}\ ..)\ (\texttt{S}\ ..))[\gamma] \to (\texttt{eq}\ (\texttt{T}\ ..)\ (\texttt{S}\ ..))[\gamma] =$

$\text{fn}\ \texttt{e} \Rightarrow \text{case}\ \texttt{e}\ \text{of}:$

Beluga Proof: `lam` Case

$$\texttt{eq\_lam} : (\Pi \texttt{x} : \texttt{exp}.\ \texttt{eq}\ \texttt{x}\ \texttt{x} \rightarrow \texttt{eq}\ (\texttt{E}\ \texttt{x})\ (\texttt{F}\ \texttt{x})) \\ \rightarrow \texttt{eq}\ (\texttt{lam}\ (\lambda x.\ \texttt{E}\ \texttt{x}))\ (\texttt{lam}\ (\lambda x.\ \texttt{F}\ \texttt{x})).$$

$| \ [\gamma]\ \texttt{e\_l}\ (\lambda \texttt{x}.\lambda \texttt{u}.\texttt{D}\ ..\ \texttt{x}\ \texttt{u}) \Rightarrow$

$\vdots$

in

$[\gamma]\ \texttt{eq\_lam}\ (\lambda \texttt{x}.\lambda \texttt{v}.\texttt{F}\ ..\ \texttt{x}\ \texttt{v})$

Beluga Proof: `lam` Case

$$\texttt{eq\_lam} : (\Pi \texttt{x} : \texttt{exp}.\ \texttt{eq}\ \texttt{x}\ \texttt{x} \to \texttt{eq}\ (\texttt{E}\ \texttt{x})\ (\texttt{F}\ \texttt{x})) \\ \to \texttt{eq}\ (\texttt{lam}\ (\lambda x.\ \texttt{E}\ \texttt{x}))\ (\texttt{lam}\ (\lambda x.\ \texttt{F}\ \texttt{x})).$$

$$\begin{array}{l} \mid\ [\gamma]\ \texttt{e\_l}\ (\lambda \texttt{x}.\lambda \texttt{u}.\texttt{D}\ ..\ \texttt{x}\ \texttt{u}) \Rightarrow \\ \quad \text{let}\ [\gamma, \texttt{b} : \texttt{block}\ \texttt{x} : \texttt{exp},\ \texttt{u} : \texttt{eq}\ \texttt{x}\ \texttt{x}.\ \texttt{equal}\ \texttt{x}\ \texttt{x}]\ \texttt{F}\ ..\ \texttt{b.1}\ \texttt{b.2} = \\ \quad\ \texttt{ceq}\ ([\gamma, \texttt{b} : \texttt{block}\ \texttt{x} : \texttt{exp},\ \texttt{u} : \texttt{eq}\ \texttt{x}\ \texttt{x}.\ \texttt{equal}\ \texttt{x}\ \texttt{x}]\ \texttt{D}\ ..\ \texttt{b.1}\ \texttt{b.3}) \\ \quad \text{in} \\ \quad\ [\gamma]\ \texttt{eq\_lam}\ (\lambda \texttt{x}.\lambda \texttt{v}.\texttt{F}\ ..\ \texttt{x}\ \texttt{v}) \end{array}$$

- Applying the ind. hyp. corresponds to the recursive call.
- The context is extended with new declarations about variables in the form specified by the context schema.

Beluga Proof: `lam` Case

```
eq_lam : (Πx : exp. eq x x → eq (E x) (F x))
           → eq (lam (λx. E x)) (lam (λx. F x)).

| [γ] e_l (λx.λu.D .. x u) ⇒
 let [γ, b : block x : exp, u : eq x x. equal x x] F .. b.1 b.2 =
  ceq ([γ, b : block x : exp, u : eq x x. equal x x] D .. b.1 b.3)
 in
  [γ] eq_lam (λx.λv.F .. x v)
```

- Applying the ind. hyp. corresponds to the recursive call.
- The context is extended with new declarations about variables in the form specified by the context schema.
- Note that `D` only depends on $\gamma, \mathtt{x} : \mathtt{exp},\ \mathtt{u} : \mathtt{equal\ x\ x}$. Weakening is built-in.

Beluga Proof: `lam` Case

$$\texttt{eq\_lam} : (\Pi \texttt{x} : \texttt{exp}.\ \texttt{eq}\ \texttt{x}\ \texttt{x} \to \texttt{eq}\ (\texttt{E}\ \texttt{x})\ (\texttt{F}\ \texttt{x})) \\ \to \texttt{eq}\ (\texttt{lam}\ (\lambda x.\ \texttt{E}\ \texttt{x}))\ (\texttt{lam}\ (\lambda x.\ \texttt{F}\ \texttt{x})).$$

$$\mid [\gamma]\ \texttt{e\_l}\ (\lambda \texttt{x}.\lambda \texttt{u}.\texttt{D}\ ..\ \texttt{x}\ \texttt{u}) \Rightarrow$$

$$\text{let}\ [\gamma, \texttt{b} : \texttt{block}\ \texttt{x} : \texttt{exp},\ \texttt{u} : \texttt{eq}\ \texttt{x}\ \texttt{x}.\ \texttt{equal}\ \texttt{x}\ \texttt{x}]\ \texttt{F}\ ..\ \texttt{b.1}\ \texttt{b.2} =$$

$$\texttt{ceq}\ ([\gamma, \texttt{b} : \texttt{block}\ \texttt{x} : \texttt{exp},\ \texttt{u} : \texttt{eq}\ \texttt{x}\ \texttt{x}.\ \texttt{equal}\ \texttt{x}\ \texttt{x}]\ \texttt{D}\ ..\ \texttt{b.1}\ \texttt{b.3})$$

$$\text{in}$$

$$[\gamma]\ \texttt{eq\_lam}\ (\lambda \texttt{x}.\lambda \texttt{v}.\texttt{F}\ ..\ \texttt{x}\ \texttt{v})$$

- Applying the ind. hyp. corresponds to the recursive call.
- The context is extended with new declarations about variables in the form specified by the context schema.
- Note that `D` only depends on $\gamma, \texttt{x} : \texttt{exp},\ \texttt{u} : \texttt{equal}\ \texttt{x}\ \texttt{x}$. Weakening is built-in.
- `F` is the result of the recursive call and only depends on $\texttt{x} : \texttt{exp}$ and $\texttt{u} : \texttt{eq}\ \texttt{x}\ \texttt{x}$; strengthening is also built-in.
- `F` is used to assemble the final result.

Hybrid Specification Logic (Intermediate Level)

- Here, we use a sequent formulation of a second-order minimal logic with backchaining as a specification logic (SL).
- Sequents: $\Gamma \rhd_n G$
  - Contexts Γ are explicit at this level.
  - Integer n indicates the height of a derivation.
  - G is a formula of the SL.
  - $\_ \rhd_{\_} \_$ is defined as an inductive predicate in Coq or Isabelle/HOL.

Using the SL to encode OL Rules

- The definition of $\_ \rhd_{\_} \_$ is parameterized by atoms A of a particular object logic (OL).
- The SL includes a backchain rule on clauses of the form $(A \longleftarrow G)$.
- $\_ \longleftarrow \_$ is also defined as an inductive predicate.
- Using these definitions, the rules of the OL are encoded in the SL in exactly the same form as in Twelf and Beluga, including hypothetical and parametric judgments.

Using the SL to encode OL Rules

- The definition of $\_ \rhd \_$ is parameterized by atoms A of a particular object logic (OL).
- The SL includes a backchain rule on clauses of the form $(A \longleftarrow G)$.
- $\_ \longleftarrow \_$ is also defined as an inductive predicate.
- Using these definitions, the rules of the OL are encoded in the SL in exactly the same form as in Twelf and Beluga, including hypothetical and parametric judgments.
- E.g.,

$$\begin{aligned} eq\_lam : \quad & \text{abstr}\, E \;\rightarrow\; \text{abstr}\, F \;\rightarrow \\ & \quad \text{eq}\ (\text{lam}\, x.\, Ex)\ (\text{lam}\, x.\, Fx) \longleftarrow \\ & \quad\quad \text{all}\, x.\, (\text{eq}\ x\ x)\ \text{imp}\ \langle \text{eq}\ (Ex)\ (Fx) \rangle \end{aligned}$$

where all and imp are connectives of the SL.

Theorem 2 (Completeness)

- Attempt 1: If $\Phi \vdash \text{equal } M\ N$ then $\Psi \vdash \text{eq } M\ N$.
- Problem: this statement does not contain enough information about how the two contexts Φ and Ψ are related.
- Solution 1: Add more info (later)
- ...

Theorem 2 (Completeness)

- Attempt 1: If $\Phi \vdash \text{equal } M\ N$ then $\Psi \vdash \text{eq } M\ N$.
- Problem: this statement does not contain enough information about how the two contexts Φ and Ψ are related.
- Solution 1: Add more info (later)
- $\ldots$

Context Invariants

- $\text{ceq\_inv } \Phi\ \Psi == (\forall x\ y.\ \text{equal } x\ y \in \Phi \to \text{eq } x\ y \in \Psi) \wedge \cdots$

Theorem 2 (Completeness)

- Attempt 1: If $\Phi \vdash \text{equal } M\ N$ then $\Psi \vdash \text{eq } M\ N$.
- Problem: this statement does not contain enough information about how the two contexts Φ and Ψ are related.
- Solution 1: Add more info (later)
- ...

Context Invariants

- $\text{ceq\_inv } \Phi\ \Psi == (\forall x\ y.\ \text{equal } x\ y \in \Phi \to \text{eq } x\ y \in \Psi) \wedge \cdots$
- Lemma (Context Extension):
  $\text{ceq\_inv } \Phi\ \Psi \to \text{ceq\_inv } (\text{equal } x\ x, \text{is\_tm } x, \Phi)\ (\text{eq } x\ x, \Psi)$

Theorem 2 (Completeness)

- Attempt 1: If $\Phi \vdash \text{equal } M\ N$ then $\Psi \vdash \text{eq } M\ N$.
- Problem: this statement does not contain enough information about how the two contexts Φ and Ψ are related.
- Solution 1: Add more info (later)
- ...

Context Invariants

- $\text{ceq\_inv } \Phi\ \Psi == (\forall x\ y.\ \text{equal } x\ y \in \Phi \to \text{eq } x\ y \in \Psi) \wedge \cdots$
- Lemma (Context Extension):
  $\text{ceq\_inv } \Phi\ \Psi \to \text{ceq\_inv } (\text{equal } x\ x, \text{is\_tm } x, \Phi)\ (\text{eq } x\ x, \Psi)$
- Theorem 2 (Completeness):
  $\text{ceq\_inv } \Phi\ \Psi \to \Phi \rhd_n \langle \text{equal } T\ S \rangle \to \Psi \rhd_n \langle \text{eq } T\ S \rangle$

Alternative Version: Generalized Contexts in Hybrid

- Theorem 2 (Completeness): $\Gamma \rhd_n \langle \text{equal } T\ S \rangle \rightarrow \Gamma \rhd_n \langle \text{eq } T\ S \rangle$
- We must explicitly define weakening and strengthening functions on Γ.
- We must explicitly state and prove weakening and strengthening lemmas.
- We must explicitly apply these lemmas.
- Much of this reasoning should be easy to automate.

Another Look at the Criteria: Contexts

- How do we represent contexts in proofs?
  - Beluga: explicit contexts whose structure is defined by context schemas
  - Twelf: implicit contexts
  - Hybrid: explicit lists or sets in the SL

How do we reason with contexts?

- Beluga and Twelf
  - built-in support for weakening and strengthening
  - supported by underlying typing rules and context subsumption
  - sensitive to ordering of elements in a block
  - may require explicit weakening

How do we reason with contexts?

- Beluga and Twelf
  - built-in support for weakening and strengthening
  - supported by underlying typing rules and context subsumption
  - sensitive to ordering of elements in a block
  - may require explicit weakening
- Hybrid:
  - weakening supported by a lemma at the SL level
  - used to reason about weakening and strengthening for each object logic
  - requires explicit weakening/strengthening lemmas
  - much of the reasoning is stereotyped and could be automated

How do we know we have implemented a proof?

- Hybrid:
  - all reasoning is explicit
  - simply need to trust the underlying proof assistant and establish adequacy
  - extensive support for induction, etc.

How do we know we have implemented a proof?

- Hybrid:
  - all reasoning is explicit
  - simply need to trust the underlying proof assistant and establish adequacy
  - extensive support for induction, etc.
- Twelf:
  - must establish separately that user implemented a total function
  - provides a coverage checker which relies on the block (and world) declarations to ensure the base cases are covered
  - a termination checker verifies that all appeals to the induction hypothesis are valid

How do we know we have implemented a proof?

- Hybrid:
  - all reasoning is explicit
  - simply need to trust the underlying proof assistant and establish adequacy
  - extensive support for induction, etc.
- Twelf:
  - must establish separately that user implemented a total function
  - provides a coverage checker which relies on the block (and world) declarations to ensure the base cases are covered
  - a termination checker verifies that all appeals to the induction hypothesis are valid
- Beluga:
  - approach is similar to Twelf
  - theoretical foundation for coverage is established; implementation is planned
  - should be able to adapt existing work on termination checking to Beluga, though not done yet

Other Benchmarks

- Equality Reasoning for Lambda-Terms
  - Theorem 2 requires:
    Theorem 1 (Admissibility of Reflexivity and Transitivity)
    1. If Ψ contains premises for all the free variables in M, then $\Psi \vdash \text{eq}\ M\ M$.
    2. If $\Psi \vdash \text{eq}\ M\ L$ and $\Psi \vdash \text{eq}\ L\ N$ then $\Psi \vdash \text{eq}\ M\ N$.

Other Benchmarks

- Equality Reasoning for Lambda-Terms
  - Theorem 2 requires:
    Theorem 1 (Admissibility of Reflexivity and Transitivity)
    1. If Ψ contains premises for all the free variables in M, then $\Psi \vdash \text{eq}\ M\ M$.
    2. If $\Psi \vdash \text{eq}\ M\ L$ and $\Psi \vdash \text{eq}\ L\ N$ then $\Psi \vdash \text{eq}\ M\ N$.
- Reasoning About Variable Occurrences
  - $\Psi \vdash \text{shape}\ M_1\ M_2$ describes when the term M_1 and the term M_2 have the same shape or structure.
  - Example theorem: If $\Psi \vdash \text{shape}\ M_1\ M_2$ then there exists an I such that $\Psi \vdash \text{var-occ}\ M_1\ I$ and $\Psi \vdash \text{var-occ}\ M_2\ I$. Furthermore I is unique.

Other Benchmarks

- Equality Reasoning for Lambda-Terms
  - Theorem 2 requires:
    Theorem 1 (Admissibility of Reflexivity and Transitivity)
    1. If Ψ contains premises for all the free variables in M, then $\Psi \vdash \text{eq}\ M\ M$.
    2. If $\Psi \vdash \text{eq}\ M\ L$ and $\Psi \vdash \text{eq}\ L\ N$ then $\Psi \vdash \text{eq}\ M\ N$.
- Reasoning About Variable Occurrences
  - $\Psi \vdash \text{shape}\ M_1\ M_2$ describes when the term M_1 and the term M_2 have the same shape or structure.
  - Example theorem: If $\Psi \vdash \text{shape}\ M_1\ M_2$ then there exists an I such that $\Psi \vdash \text{var-occ}\ M_1\ I$ and $\Psi \vdash \text{var-occ}\ M_2\ I$. Furthermore I is unique.
- Reasoning About Subterms in Lambda-Terms
  - $\Psi \vdash \text{le}\ M\ N$ describes when a given lambda-term M is a subterm of (structurally smaller than) another lambda-term N.
  - Theorem: If for all N. $\Psi \vdash \text{le}\ N\ K$ implies $\Psi \vdash \text{le}\ N\ L$ then $\Psi \vdash \text{le}\ K\ L$.

Conclusion

- We present benchmark problems, together with a general set of criteria for comparing reasoning systems that support HOAS.
- We compare proofs of one of these problems in three systems (Beluga, Twelf, and Hybrid) using our criteria.
- See complogic.cs.mcgill.ca/beluga/benchmarks/.
- This work is a starting point that:
  - will help users and developers evaluate proof assistants for reasoning about meta-theory of programming languages;
  - can facilitate a better understanding of the differences between and limitations of these systems;
  - can help with understanding the impact of these design decisions in practice;
  - can provide guidance for users and stimulate discussion among developers.
- Future work includes implementing these benchmarks in other systems such as Abella and Delphin.

Beluga Proof of Theorem 2

- Context schemas classify contexts:

  $\text{schema}\ \texttt{eCtx} = \texttt{block}\ \texttt{x} : \texttt{exp},\ \texttt{u} : \texttt{eq}\ \texttt{x}\ \texttt{x}.\ \texttt{equal}\ \texttt{x}\ \texttt{x}$

- Inductive proofs about derivations are written as recursive functions using pattern matching.

$\text{rec}\ \texttt{ceq} : \{\gamma : \texttt{eCtx}\}(\texttt{equal}\ (\texttt{T}\ ..)\ (\texttt{S}\ ..))[\gamma] \to (\texttt{eq}\ (\texttt{T}\ ..)\ (\texttt{S}\ ..))[\gamma] =$
$\text{fn}\ \texttt{e} \Rightarrow \text{case}\ \texttt{e}\ \text{of}$

$\vdots$

Beluga Proof of Theorem 2

- Context schemas classify contexts:

  $\text{schema}\ \texttt{eCtx} = \texttt{block}\ \texttt{x} : \texttt{exp},\ \texttt{u} : \texttt{eq}\ \texttt{x}\ \texttt{x}.\ \texttt{equal}\ \texttt{x}\ \texttt{x}$

- Inductive proofs about derivations are written as recursive functions using pattern matching.

$\text{rec}\ \texttt{ceq} : \{\gamma : \texttt{eCtx}\}(\texttt{equal}\ (\texttt{T}\ ..)\ (\texttt{S}\ ..))[\gamma] \to (\texttt{eq}\ (\texttt{T}\ ..)\ (\texttt{S}\ ..))[\gamma] =$
$\text{fn}\ \texttt{e} \Rightarrow \text{case}\ \texttt{e}\ \text{of}$
$\mid [\gamma]\ \#\texttt{p.3}\ .. \Rightarrow [\gamma]\ \#\texttt{p.2}\ ..$
$\vdots$

- The case where the proof is by assumption from context is modelled using a parameter variable $\#\texttt{p}$, which represents a block.

Hybrid Specification Logic (Intermediate Level)

- Here, we use a sequent formulation of a second-order minimal logic with backchaining as a specification logic (SL).
- Sequents: $\Gamma \rhd_n G$
  - Contexts Γ are explicit at this level.
  - Integer n indicates the height of a derivation.
  - G is a formula of the SL.
  - Parameterized by atoms A of a particular object logic (OL).
  - Notation: $\langle A \rangle$. The brackets coerce OL atoms to SL formulas.

Hybrid Specification Logic (Intermediate Level)

- Here, we use a sequent formulation of a second-order minimal logic with backchaining as a specification logic (SL).
- Sequents: $\Gamma \rhd_n G$
  - Contexts Γ are explicit at this level.
  - Integer n indicates the height of a derivation.
  - G is a formula of the SL.
  - Parameterized by atoms A of a particular object logic (OL).
  - Notation: $\langle A \rangle$. The brackets coerce OL atoms to SL formulas.
- $\_ \rhd_\_ \_$ is defined as an inductive predicate in Coq or Isabelle/HOL. Example clauses include:
  - And-intro: $\Gamma \rhd_n G_1 \rightarrow \Gamma \rhd_n G_2 \rightarrow \Gamma \rhd_{n+1} (G_1 \text{ and } G_2)$
  - Initial: $(A \in \Gamma) \rightarrow \Gamma \rhd_n \langle A \rangle$
  - Backchain on OL clause: $(A \longleftarrow G) \rightarrow \Gamma \rhd_n G \rightarrow \Gamma \rhd_{n+1} \langle A \rangle$

Inference Rules of the Object Logic (Again)

$$\frac{\Psi, \mathsf{eq}\ x\ x \vdash \mathsf{eq}\ M\ N}{\Psi \vdash \mathsf{eq}\ (\mathsf{lam}\, x.\, M)\ (\mathsf{lam}\, x.\, N)}$$

Hybrid Encoding

$\_ \longleftarrow \_$ is defined as an inductive predicate in Coq or Isabelle/HOL. For example:

$$\begin{aligned} \mathit{eq\_lam}: \quad & \mathsf{abstr}\, E \ \rightarrow\ \mathsf{abstr}\, F \ \rightarrow \\ & \mathsf{eq}\ (\mathsf{lam}\, x.\, Ex)\ (\mathsf{lam}\, x.\, Fx) \longleftarrow \\ & \quad \mathsf{all}\, x.\, (\mathsf{eq}\ x\ x)\ \mathsf{imp}\ \langle \mathsf{eq}\ (Ex)\ (Fx) \rangle \end{aligned}$$

where all and imp are connectives of the SL.

Another Look at the Criteria: Contexts

- How do we retrieve elements from a context?
  - Beluga: supported by parameter variables and projections
  - Twelf:
    - ⋆ no access since context is implicit
    - ⋆ base cases are handled when an assumption is introduced
    - ⋆ may lead to scattering of base cases and redundancy
  - Hybrid: via simple list or set operations such as membership