Programming logical relations proofs

Brigitte Pientka
School of Computer Science
McGill University
Montreal, Canada

Joint work with Andrew Cave
Motivation

How to program and reason with formal systems and proofs?

- Formal systems (given via axioms and inference rules) play an important role when designing languages and more generally software.

- Proofs (that a given property is satisfied) are an integral part of the software.
Motivation

How to program and reason with formal systems and proofs?

• Formal systems (given via axioms and inference rules) play an important role when designing languages and more generally software.

• Proofs (that a given property is satisfied) are an integral part of the software.

What are good meta-languages to program and reason with formal systems and proofs?
This Talk

Design and implementation of Beluga

- Introduction
- Example: Proof by logical relation
- Writing a proof in Beluga . . .
- Conclusion and current work

“The limits of my language mean the limits of my world.”

- L. Wittgenstein
This Talk

Design and implementation of Beluga

- Introduction
- Example: Proof by logical relations
- Writing a proof in Beluga . . .
- Conclusion and current work

“The limits of my language mean the limits of my world.”
- L. Wittgenstein
Simply Typed Lambda-calculus (Gentzen-style)

Types and Terms

Types $A, B ::= i \mid A \rightarrow B$

Terms $M, N ::= x \mid c \mid \text{lam } x.M \mid \text{app } M N$

Evaluation Judgment:

$$M \rightarrow M'$$

read as “$M$ steps to $M'$”

- $\text{app } (\text{lam } x.M) N \rightarrow [N/x]M$ (s/beta)
- $M \rightarrow M$ (s/refl)
- $M \rightarrow M' \rightarrow N$ (s/trans)
- $\text{app } M N \rightarrow \text{app } M' N$ (s/app)
Simply Typed Lambda-calculus (Gentzen-style)

Types and Terms

Types $A, B ::= \text{i}$

Terms $M, N ::= x \mid c \mid A \rightarrow B \mid \text{lam} \ x. M \mid \text{app} \ M \ N$

Evaluation Judgment: $M \rightarrow M'$ read as “$M$ steps to $M'$”

- $\text{app} \ (\text{lam} \ x. M) \ N \rightarrow [N/x]M$ s/beta
- $M \rightarrow M'$ s/app
- $\text{app} \ M \ N \rightarrow \text{app} \ M' \ N$ s/trans

Typing Judgment: $M : A$ read as “$M$ has type $A$” (Gentzen-style)

- $x : A \ u$
- $\text{const} \ c : \text{i}$
- $(\text{lam} \ x. M) : (A \rightarrow B)$ lam$^{x,u}$
- $(\text{app} \ M \ N) : B$ app

- $M : (A \rightarrow B) \ N : A$
Simply Typed Lambda-calculus with Contexts

Types and Terms

Types $A, B ::= \ i | A \to B$

Terms $M, N ::= x | c | \text{lam } x. M | \text{app } M N$

Evaluation Judgment: $\frac{}{M \to M'}$, read as “$M$ steps to $M'$”

$s/beta$:

$\frac{}{\text{app } (\text{lam } x. M) N \to [N/x]M}$

$s/\text{refl}$:

$\frac{}{M \to M}$

$s/\text{app}$:

$\frac{}{\text{app } M N \to \text{app } M' N}$

$s/\text{trans}$:

$\frac{}{M \to M' \quad M' \to N \quad M \to N}$

Typing Judgment: $\frac{}{\Gamma \vdash M : A}$, read as “$M$ has type $A$ in context $\Gamma$”

$x : A \in \Gamma$  

$\frac{}{\Gamma, x : A \vdash M : B}$  

$\frac{}{\text{lam } x \quad \Gamma \vdash M : (A \to B) \quad \Gamma \vdash N A \quad \Gamma \vdash (\text{app } M N) : B}$

Context $\Gamma ::= \cdot \mid \Gamma, x : A$  

We are introducing the variable $x$ together with the assumption $x : A$
Talking about Derivations

Typing rules

\[
\frac{x : A \in \Gamma}{\Gamma \vdash x : A}
\quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\text{lam } x. M) : (A \to B)}
\]

Evaluation rules

\[
\frac{\text{app } (\text{lam } x.M) N \longrightarrow [N/x]M}{\text{beta}}
\quad \frac{M \longrightarrow M'}{\text{app}}
\quad \frac{\text{app } M N \longrightarrow \text{app } M' N}{\text{app}}
\]
Talking about Derivations

Typing rules

\[ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\text{lam} \ x. M) : (A \rightarrow B)} \]

Evaluation rules

\[ \frac{\Gamma \vdash M : (A \rightarrow B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\text{app} \ M \ N) : B} \]

\[ \frac{(\text{lam} \ x. M) \ N \rightarrow [N/x]M}{\text{app}} \]

\[ M \rightarrow M' \quad \frac{\text{app} \ M \ N \rightarrow \text{app} \ M' \ N}{\text{app}} \]

- What kinds of variables are used?

Bound variables, Schematic variables

Meta-variables, Parameter variables, Context variables

Substitution for bound variable, Renaming of bound variables, Substitution for schematic variables

Structured sequences, Uniqueness of declaration, Weakening, Substitution lemma, etc.
Talking about Derivations

Typing rules

\[
\begin{align*}
\Gamma, x : A \vdash \Gamma \vdash x : A \\
\Gamma, x : A \vdash M : B \quad \Gamma \vdash (\text{lam } x. M) : (A \to B) \\
\Gamma \vdash \Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B \quad \Gamma \vdash (\text{app } M \ N) : B
\end{align*}
\]

Evaluation rules

\[
\begin{align*}
\text{beta} \quad \text{app} \\
\text{beta} \quad \text{app} \\
\text{app } (\text{lam } x. M) \ N \longrightarrow [N/x] M \\
\text{app } M \ N \longrightarrow \text{app } M' \ N
\end{align*}
\]

• What kinds of variables are used? **Bound variables, Schematic variables**
  in particular: **Meta-variables, Parameter variables, Context variables**
Talking about Derivations

Typing rules

\[
\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\text{lam } x. M) : (A \rightarrow B)} \quad \frac{\Gamma \vdash M : (A \rightarrow B)}{\Gamma \vdash \text{app } M N : B}
\]

Evaluation rules

\[
\frac{\text{app } (\text{lam } x. M) N \rightarrow [N/x]M}{\text{beta}} \quad \frac{M \rightarrow M'}{\text{app } M N \rightarrow \text{app } M' N} \quad \frac{M \rightarrow M'}{\text{app } M N \rightarrow \text{app } M' N} \quad \frac{M \rightarrow M'}{\text{app } M N \rightarrow \text{app } M' N}
\]

- What kinds of variables are used? **Bound variables, Schematic variables**
  - in particular: Meta-variables, Parameter variables, Context variables
- What operations on variables are needed?
Talking about Derivations

Typing rules
\[
\begin{align*}
x : A & \in \Gamma \\
\Gamma & \vdash x : A \\
\Gamma, x : A & \vdash M : B \\
\Gamma & \vdash (\text{lam } x. M) : (A \rightarrow B)
\end{align*}
\]
\[
\begin{align*}
\Gamma & \vdash M : (A \rightarrow B) \\
\Gamma & \vdash N : B \\
\Gamma & \vdash (\text{app } M \ N) : B
\end{align*}
\]

Evaluation rules
\[
\begin{align*}
\text{app} (\text{lam } x. M) \ N & \rightarrow [N/x]M \\
\text{beta} & \\
\text{app} \ M \ N & \rightarrow \text{app} \ M' \ N
\end{align*}
\]

• What kinds of variables are used? **Bound variables, Schematic variables** in particular: Meta-variables, Parameter variables, Context variables

• What operations on variables are needed? **Substitution for bound variable**, **Renaming of bound variables**, **Substitution for schematic variables**
Talking about Derivations

Typing rules

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$
$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\text{lam} \ x. M) : (A \rightarrow B)}$$
$$\frac{\Gamma \vdash M : (A \rightarrow B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\text{app} \ M \ N) : B}$$

Evaluation rules

$$\frac{\text{app} \ (\text{lam} \ x. M) \ N \rightarrow [N/x]M}{\beta}$$
$$\frac{M \rightarrow M'}{\text{app} \ M \ N \rightarrow \text{app} \ M' \ N}$$

- What kinds of variables are used? **Bound variables, Schematic variables**
  - in particular: **Meta-variables, Parameter variables, Context variables**
- What operations on variables are needed? **Substitution for bound variable, Renaming of bound variables, Substitution for schematic variables**
- How should we represent contexts? **What properties do contexts have?**
Talking about Derivations

Typing rules

\[ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \]
\[ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\text{lam} \ x. M) : (A \to B)} \]
\[ \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\text{app} \ M \ N) : B} \]

Evaluation rules

\[ \frac{\text{app} (\text{lam} \ x. M) \ N \rightarrow [N/x]M}{\beta} \]
\[ \frac{M \rightarrow M'}{\text{app} \ M \ N \rightarrow \text{app} \ M' \ N} \]

- What kinds of variables are used? **Bound variables, Schematic variables** in particular: Meta-variables, Parameter variables, Context variables
- What operations on variables are needed? **Substitution for bound variable, Renaming of bound variables, Substitution for schematic variables**
- How should we represent contexts? **What properties do contexts have?** (Structured) sequences, Uniqueness of declaration, Weakening, Substitution lemma, etc.
Talking about Derivations

Typing rules

\[
\begin{align*}
\frac{x : A \in \Gamma}{\Gamma \vdash x : A} & \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\text{lam } x. M) : (A \to B)} \\
\end{align*}
\]

\[
\begin{align*}
\frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\text{app } M \ N) : B} & \quad \frac{\Gamma \vdash M : (A \to B)}{\Gamma \vdash M : (A \to B)} \\
\end{align*}
\]

Evaluation rules

\[
\begin{align*}
\frac{\text{app } (\text{lam } x. M) \ N \to [N/x]M}{\text{beta}} & \quad \frac{M \to M'}{\text{app } M \ N \to \text{app } M' \ N} \\
\end{align*}
\]

- What kinds of variables are used? Bound variables, Schematic variables in particular: Meta-variables, Parameter variables, Context variables

- What operations on variables are needed? Substitution for bound variable, Renaming of bound variables, Substitution for schematic variables

- How should we represent contexts? What properties do contexts have? (Structured) sequences, Uniqueness of declaration, Weakening, Substitution lemma, etc.

Any mechanization of proofs must deal with these issues; it is just a matter how much support one gets in a given meta-language.
Weak Normalization for Simply Typed Lambda-calculus

Theorem
If $\vdash M : A$ then $M$ halts, i.e. there exists a value $V$ s.t. $M \rightarrow^* V$.

Proof.
1. Define reducibility candidate
   $R_A \triangleright B = \{ M | M \text{ halts and } \forall N \in R_A, (\text{app } M N) \in R_B \}$
2. If $M \in R_A$ then $M$ halts.
3. Backwards closed: If $M' \in R_A$ and $M \rightarrow M'$ then $M \in R_A$.
4. Fundamental Lemma: If $\vdash M : A$ then $M \in R_A$. (Requires a generalization)
Theorem

If $\vdash M : A$ then $M$ halts, i.e. there exists a value $V$ s.t. $M \rightarrow^* V$. 

Proof.

1. Define reducibility candidate $R_A R_i = \{M | M \text{ halts} \}$

2. $R_A R_B = \{M | M \text{ halts and } \forall N \in R_A, (\text{app } M N) \in R_B \}$

3. If $M \in R_A$ then $M$ halts.

4. Backwards closed: If $M' \in R_A$ and $M \rightarrow M'$ then $M \in R_A$.

5. Fundamental Lemma: If $\vdash M : A$ then $M \in R_A$. (Requires a generalization)
Weak Normalization for Simply Typed Lambda-calculus

**Theorem**

If $\vdash M : A$ then $M$ halts, i.e. there exists a value $V$ s.t. $M \rightarrow^* V$.

**Proof.**

1. Define reducibility candidate $\mathcal{R}_A$
   
   \[
   \begin{align*}
   \mathcal{R}_i & = \{M \mid M \text{ halts}\} \\
   \mathcal{R}_{A\rightarrow B} & = \{M \mid M \text{ halts and } \forall N \in \mathcal{R}_A, (\text{app } M N) \in \mathcal{R}_B\}
   \end{align*}
   \]

2. If $M \in \mathcal{R}_A$ then $M$ halts.

3. Backwards closed: If $M' \in \mathcal{R}_A$ and $M \rightarrow M'$ then $M \in \mathcal{R}_A$.

4. **Fundamental Lemma:** If $\vdash M : A$ then $M \in \mathcal{R}_A$. (Requires a generalization)
Generalization of Fundamental Lemma

Lemma (Main lemma)

If $D : \Gamma \vdash M : A$ and $\sigma \in R_\Gamma$ then $[\sigma]M \in R_A$.

where $\sigma \in R_\Gamma$ is defined as:

$\cdot \in R.
\frac{\sigma \in R_\Gamma\quad N \in R_A}{(\sigma, N/x) \in R_{\Gamma,x:A}}$
Generalization of Fundamental Lemma

Lemma (Main lemma)

If \( D : \Gamma \vdash M : A \) and \( \sigma \in R_\Gamma \) then \( [\sigma]M \in R_A \).
Generalization of Fundamental Lemma

Lemma (Main lemma)

\[ \text{If } \mathcal{D} : \Gamma \vdash M : A \text{ and } \sigma \in \mathcal{R}_\Gamma \text{ then } [\sigma]M \in \mathcal{R}_A. \]

Proof.

\textbf{Case } \mathcal{D} = \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \text{ var}

\sigma \in \mathcal{R}_\Gamma

[\sigma](x) = M \in \mathcal{R}_A

by assumption

by lookup in \( \sigma \in \mathcal{R}_\Gamma \) and substitution property
**Generalization of Fundamental Lemma**

**Lemma (Main lemma)**

If $D : \Gamma \vdash M : A$ and $\sigma \in \mathcal{R}_\Gamma$ then $[\sigma]M \in \mathcal{R}_A$.

**Proof.**

**Case** $D = \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$ \hspace{1cm} \text{var}

$\sigma \in \mathcal{R}_\Gamma$

$[\sigma](x) = M \in \mathcal{R}_A$ \hspace{1cm} \text{by assumption}

$\sigma \in \mathcal{R}_\Gamma$

$[\sigma](x) = M \in \mathcal{R}_A$ \hspace{1cm} \text{by lookup in } \sigma \in \mathcal{R}_\Gamma \text{ and substitution property}

**Case** $D = \frac{\frac{\Gamma \vdash M : A \rightarrow B}{\Gamma \vdash \text{app} \ M \ N : B}} {\frac{\Gamma \vdash N : A}{\text{app}}}$ \hspace{1cm} \text{app}

$\sigma \in \mathcal{R}_\Gamma$

$N \in \mathcal{R}_A$

$M \in \mathcal{R}_{A \rightarrow B}$

$M \text{ halts and } \forall N' \in \mathcal{R}_A. \ (\text{app} \ M \ N') \in \mathcal{R}_B$

$\text{app} \ M \ N \in \mathcal{R}_B$ \hspace{1cm} \text{by previous lines (}\forall\text{-elim)}

by i.h. $D_2$

by i.h. $D_1$

by definition
Generalization of Fundamental Lemma

Lemma (Main lemma)

If $D : \Gamma \vdash M : A$ and $\sigma \in R_{\Gamma}$ then $[\sigma]M \in R_A$.

Proof.

Case $D = D_1$

$$
\begin{align*}
\Gamma & \vdash M : B \\
\Gamma & \vdash \text{lam } x.M : A \rightarrow B
\end{align*}
$$

by properties of substitution

[\sigma](\text{lam } x.M) = \text{lam } x.([\sigma, x/x]M)

by definition

halts (\text{lam } x.[\sigma, x/x]M)

by I.H. on $D_1$ since $\sigma \in R_{\Gamma}$

Suppose $N \in R_A$.

$$
\begin{align*}
[\sigma, N/x]M & \in R_B \\
[N/x][\sigma, x/x]M & \in R_B \\
\text{app } (\text{lam } x. [\sigma, x/x]M) & N \in R_B
\end{align*}
$$

by properties of substitution

by Backwards closure

Hence $[\sigma](\text{lam } x.M) \in R_{A \rightarrow B}$

by definition
Challenging Benchmark

- Model different level of bindings
  lambda-binder, $\forall$ in reducibility definition $R$, quantification over substitutions and contexts

- Simultaneous substitution and algebraic properties
  Substitution lemma, Reason about composition, decomposition, associativity, identity, etc.
  
  \[
  [\cdot]M = M
  \]
  
  \[
  [\sigma, N/x]M = [N/x][\sigma, x/x]M
  \]
  
  \[
  [\sigma_1][\sigma_2]M = [[\sigma_1]\sigma_2]M
  \]

  A dozen such properties are needed

- Main known approaches
  - Coq/Agda lack support for substitutions and binders
  - Twelf, Delphin are too weak (to do it directly)
  - Abella allows normalization proofs but lacks support for contexts; still need to implement some substitution/context properties
Introduction

This Talk

Design and implementation of Beluga

- Introduction
- Example: Proof by logical relations
- Writing a proof in Beluga ...
- Conclusion and current work
Beluga$^\mu$: Two Level Approach

Level 1: Contextual logical framework LF [HHP’93, TOCL’08]

- Compact representation of formal systems and derivations
- Higher-order abstract syntax and dependent types
Beluga$\mu$: Two Level Approach

Level 1: Contextual logical framework LF [HHP’93, TOCL’08]

- Compact representation of formal systems and derivations
- Higher-order abstract syntax and dependent types
  $\leadsto$ support for $\alpha$-renaming, substitution, adequate representations
- Contextual LF: Contextual types characterize contextual objects [TOCL’08]
  $\leadsto$ support well-scoped derivations
  $\leadsto$ abstract notion of contexts and substitution [POPL’08, LFMTP’13]
Beluga$\mu$: Two Level Approach

Level 1: Contextual logical framework LF [HHP’93,TOCL’08]
- Compact representation of formal systems and derivations
- Higher-order abstract syntax and dependent types
  ⇝ support for $\alpha$-renaming, substitution, adequate representations
- Contextual LF: Contextual types characterize contextual objects [TOCL’08]
  ⇝ support well-scoped derivations
  ⇝ abstract notion of contexts and substitution [POPL’08,LFMTP’13]

Level 2: Functional programming with indexed types [POPL’08,POPL’12]
Proof term language for first-order logic over a specific domain (= contextual LF) together with domain-specific induction principle and recursive definitions (= indexed recursive types)
Beluga\(\mu\): Two Level Approach

**Level 1:** Contextual logical framework LF [HHP’93, TOCL’08]
- Compact representation of formal systems and derivations
- Higher-order abstract syntax and dependent types
  \(\leadsto\) support for \(\alpha\)-renaming, substitution, adequate representations
- Contextual LF: Contextual types characterize contextual objects [TOCL’08]
  \(\leadsto\) support well-scoped derivations
  \(\leadsto\) abstract notion of contexts and substitution [POPL’08, LFMTP’13]

**Level 2:** Functional programming with indexed types [POPL’08, POPL’12]

Proof term language for first-order logic over a specific domain (= contextual LF) together with domain-specific induction principle and recursive definitions (= indexed recursive types)

<table>
<thead>
<tr>
<th>On paper proof</th>
<th>Proofs as functions in Beluga</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case analysis</td>
<td>Case analysis and pattern matching</td>
</tr>
<tr>
<td>Inversion</td>
<td>Pattern matching using let-expression</td>
</tr>
<tr>
<td>Induction Hypothesis</td>
<td>Recursive call</td>
</tr>
</tbody>
</table>
Step 1: Represent Types and Lambda-terms in LF

Types and Terms

Types $A, B ::= i$ $| A \rightarrow B$

Terms $M, N ::= x | c | \text{lam} \; x. \; M | \text{app} \; M \; N$

Typing rules

\[
\begin{align*}
\text{const} & \quad \frac{x : A \quad u}{M : B} \quad \text{lam}^x \quad \frac{M : (A \rightarrow B) \quad N : A}{(\text{app} \; M \; N) : B}
\end{align*}
\]

LF representation in Beluga

```
datatype tp: type =
| i: tp
| arr: tp \rightarrow tp \rightarrow tp;

datatype tm: tp \rightarrow type =
| c : tm i
| lam: (tm A \rightarrow tm B) \rightarrow tm (arr A B)
| app: tm (arr A B) \rightarrow tm A \rightarrow tm B;
```
Step 1: Represent Types and Lambda-terms in LF

Types and Terms

Types A, B ::= i
      | A → B

Terms M, N ::= x | c
      | lam x.M
      | app M N

Typing rules

\[
\begin{align*}
\Gamma & \vdash x : A \\
\Gamma & \vdash M : B \\
\Gamma & \vdash (\text{lam } x.M) : (A \rightarrow B)
\end{align*}
\]

LF representation in Beluga

```
datatype tp: type =
| i: tp
| arr: tp → tp → tp;
```

```
datatype tm: tp → type =
| c : tm i
| lam: (tm A → tm B) → tm (arr A B)
| app: tm (arr A B) → tm A → tm B;
```
Reducibility Candidates as Indexed Types

Reducibility candidates for terms $M \in \mathcal{R}_A$:

$$\mathcal{R}_i = \{ M \mid \text{halts } M \}$$

$$\mathcal{R}_{A \rightarrow B} = \{ M \mid \text{halts } M \text{ and } \forall N \in \mathcal{R}_A, (\text{app } M N) \in \mathcal{R}_B \}$$
Reducibility Candidates as Indexed Types

Reducibility candidates for terms $M \in \mathcal{R}_A$:

- $\mathcal{R}_i = \{M \mid \text{halts } M\}$
- $\mathcal{R}_{A \rightarrow B} = \{M \mid \text{halts } M \text{ and } \forall N \in \mathcal{R}_A, (\text{app } M N) \in \mathcal{R}_B\}$

Computation-level data types in Beluga

datatype Reduce : \{A:\[ \vdash \text{tp}\]} \{M:\[ \vdash \text{tm } A]\} \text{ ctype =}
| I : \[ \vdash \text{halts } M\] \rightarrow \text{Reduce [ \vdash i] [\vdash M]} |
| Arr : \[ \vdash \text{halts } M\] \rightarrow
  (\{N:\[ \vdash \text{tm } A]\} \text{ Reduce [ \vdash A] [ \vdash N] \rightarrow \text{Reduce [ \vdash B] [ \vdash \text{app } M N]}\})
  \rightarrow \text{Reduce [ \vdash arr A B] [\vdash M]}; |

- $[\vdash \text{app } M N]$ and $[\vdash \text{arr } A B]$ are contextual types [TOCL’08].
- Note: $\rightarrow$ is overloaded.
  - $\rightarrow$ is the LF function space: binders in the object language are modelled by LF functions (used inside $[\ ]$)
  - $\rightarrow$ is a computation-level function (used outside $[\ ]$)
- Not strictly positive definition, but stratified.
Reducibility Candidates as Indexed Types

Reducibility candidates for substitutions $\sigma \in \mathcal{R}_\Gamma$:

\[
\begin{align*}
\cdot & \in \mathcal{R}. \\
\sigma & \in \mathcal{R}_\Gamma \\
N & \in \mathcal{R}_A \\
(\sigma, N/x) & \in \mathcal{R}_{\Gamma, x:A}
\end{align*}
\]
Reducibility Candidates as Indexed Types

Reducibility candidates for substitutions $\sigma \in \mathcal{R}_\Gamma$:

\[
\begin{align*}
\cdot \in \mathcal{R}. & \\
\sigma \in \mathcal{R}_\Gamma & \quad N \in \mathcal{R}_A \\
(\sigma, N/x) \in \mathcal{R}_{\Gamma, x:A} & 
\end{align*}
\]

Computation-level data types in Beluga

```haskell
datatype RedSub : (\Gamma : ctx){\sigma : \vdash \Gamma} 
ctype =
| Nil : RedSub [ \vdash ^ ]
| Cons : RedSub [ \vdash \sigma ] \rightarrow \text{Reduce} [ \vdash A ] [ \vdash M ] \rightarrow \text{RedSub} [ \vdash \sigma M ];
```

- Contexts are structured sequences and are classified by context schemas
  
  schema ctx = x : tm A.

- Substitution $\tau$ are first-class and have type $\Psi \vdash \Phi$ providing a mapping from $\Phi$ to $\Psi$.  

Theorems as Computation-level Types

Lemma (Backward closed)

If $M \rightarrow M'$ and $M' \in R_A$ then $M \in R_A$.

```
rec closed : \[ |- mstep M M' \] \rightarrow \text{Reduce} \[ |- A \] \[ |- M' \] \rightarrow \text{Reduce} \[ |- A \] \[ |- M \] = ? ;
```

Lemma (Main lemma)

If $\Gamma \vdash M : A$ and $\sigma \in R_\Gamma$ then $[\sigma]M \in R_A$.

```
rec main : \{ \Gamma : \text{ctx} \} \{ M : [\Gamma \vdash \text{tm} A] \} \text{RedSub} \[ |- \sigma \] \rightarrow \text{Reduce} \[ |- A \] \[ |- M \sigma \] = ? ;
```
Fundamental Lemma

rec closed : 
\[\vdash m\text{step } M M' \rightarrow\]
Reduce 
\[\vdash A\]
\[\vdash M' \rightarrow\]
Reduce 
\[\vdash A\]
\[\vdash M = ? ;\]

rec main :

\[
\text{\Gamma} \vdash \text{tm } A
\]

RedSub 
\[\vdash \sigma\]

\[\vdash A\]
\[\vdash M \sigma\] =

mlam \[
\text{\Gamma} 

\Rightarrow mlam M \Rightarrow fn rs 

\Rightarrow case \[
\text{\Gamma} \vdash M ...
\]
of

| \[
\text{\Gamma} \vdash \#p ...
\]
| \[
\text{\Gamma} \vdash \text{app } (M1 ... ) (M2 ... )
\]| \[
\text{\Gamma} \vdash \text{lam } (\lambda x. M1 ... x)
\] 

\| \| 

let Arr ha f = main \[
\Gamma
\]
\[
\Gamma \vdash M1 ...
\]
rs 

in

f \[
\vdash _
\]
(main \[
\Gamma
\]
\[
\Gamma \vdash M2 ...
\]
rs)

\[
mlam N \Rightarrow fn rN \Rightarrow closed \[
\vdash s/beta
\]

(main \[
\Gamma, x:tm _
\]
\[
\Gamma, x \vdash M1 ...
\]
Cons rs rN))

\[
\text{\Gamma} \vdash c \Rightarrow I \[
\vdash h/value s/refl v/c
\]; 

\|

• Direct encoding of on-paper proof

• Equations about substitution properties automatically discharged

(amounts to roughly a dozen lemmas about substitution and weakening)

• Total encoding about 75 lines of Beluga code
**Fundamental Lemma**

```plaintext
rec closed : [ ⊢ mstep M M'] → Reduce [ ⊢ A] [ ⊢ M'] → Reduce [ ⊢ A] [ ⊢ M] = ? ;
rec main : {Γ:ctx}{M:[Γ ⊢ tm A]} RedSub [ ⊢ σ] → Reduce [ ⊢ A] [ ⊢ M σ] =
```
Introduction
Beluga: Design and implementation

Fundamental Lemma

\[ \text{rec } \text{closed} : \left[ \vdash \text{mstep } M \ M' \right] \rightarrow \text{Reduce} \left[ \vdash \text{A} \right] \left[ \vdash M' \right] \rightarrow \text{Reduce} \left[ \vdash \text{A} \right] \left[ \vdash M \right] = ? ; \]

\[ \text{rec } \text{main} : \{\Gamma : \text{ctx}\}\{M : \left[ \Gamma \vdash \text{tm } A \right]\} \text{ RedSub} \left[ \vdash \sigma \right] \rightarrow \text{Reduce} \left[ \vdash \text{A} \right] \left[ \vdash M \sigma \right] = \]

\[ \text{mlam } \Gamma \Rightarrow \text{mlam } M \Rightarrow \text{fn } rs \Rightarrow \text{case } \left[ \Gamma \vdash M \ldots \right] \text{ of} \]

\[ | \left[ \Gamma \vdash \#p \ldots \right] \Rightarrow \text{lookup} \left[ \Gamma \right] \left[ \Gamma \vdash \#p \ldots \right] rs \quad \% \text{ Variable} \]
Fundamental Lemma

\[
\text{rec } \text{closed} : \left[ \vdash \text{mstep } M \; M' \right] \to \text{Reduce } \left[ \vdash A \right] \left[ \vdash M' \right] \to \text{Reduce } \left[ \vdash A \right] \left[ \vdash M \right] = ? ;
\]

\[
\text{rec } \text{main} : \{ \Gamma \text{-ctx}\}{M: \left[ \Gamma \vdash \text{tm } A \right]} \text{ RedSub } \left[ \vdash \sigma \right] \to \text{Reduce } \left[ \vdash A \right] \left[ \vdash M \sigma \right] =
\]

\[
\text{mlam } \Gamma \to \text{mlam } M \to \text{fn } \; \text{rs } \Rightarrow \text{case } \left[ \Gamma \vdash M \ldots \right] \text{ of }
\]

\[
| \left[ \Gamma \vdash \#p \ldots \right] \Rightarrow \text{lookup } \left[ \Gamma \right] \left[ \Gamma \vdash \#p \ldots \right] \; \text{rs} \quad \% \text{ Variable}
\]

\[
| \left[ \Gamma \vdash \text{app } (M1 \ldots) (M2 \ldots) \right] \Rightarrow
\]

\[
\text{let } \text{Arr } \text{ha } f = \text{main } \left[ \Gamma \right] \left[ \Gamma \vdash M1 \ldots \right] \; \text{rs } \text{in}
\]

\[
f \left[ \vdash _{\ldots} \right] \left( \text{main } \left[ \Gamma \right] \left[ \Gamma \vdash M2 \ldots \right] \; \text{rs} \right)
\]

\%

\%

\%

\%

\%

\%

\%
Fundamental Lemma

rec closed : [ ⊢ mstep M M’] → Reduce [ ⊢ A] [ ⊢ M’] →Reduce [ ⊢ A] [ ⊢ M] = ? ;
rec main : {Γ:ctx}{M:[Γ ⊢ tm A]} RedSub [ ⊢ σ] →Reduce [ ⊢ A] [ ⊢ M σ] =

mlam Γ⇒mlam M ⇒ fn rs ⇒ case [Γ ⊢ M…] of

| [Γ ⊢ #p …] ⇒ lookup [Γ] [Γ ⊢ #p …] rs % Variable

| [Γ ⊢ app (M1 …) (M2 …)] ⇒
  let Arr ha f = main [Γ] [Γ ⊢ M1 …] rs in
  f [ ⊢ _ ] (main [Γ] [Γ ⊢ M2 …] rs) % Application

| [Γ ⊢ lam (λx. M1 … x)] ⇒
  Arr [ ⊢ h/value s/refl v/lam]
  (mlam N ⇒ fn rN ⇒ closed [ ⊢ s/beta]
   (main [Γ,x:tm _] [Γ,x ⊢ M1 … x] (Cons rs rN))) % Abstraction
Fundamental Lemma

\[
\text{rec closed : } [\vdash \text{mstep} \; M \; M'] \to \text{Reduce} [\vdash A] [\vdash M'] \to \text{Reduce} [\vdash A] [\vdash M] = \star ;
\]
\[
\text{rec main : } \{\Gamma : \text{ctx}\}{M : [\Gamma \vdash \text{tm} \; A]} \; \text{RedSub} [\vdash \sigma] \to \text{Reduce} [\vdash A] [\vdash M \; \sigma] =
\]
\[
\text{mlam } \Gamma \Rightarrow \text{mlam } M \Rightarrow \text{fn } \; \text{rs} \Rightarrow \text{case } [\Gamma \vdash M \ldots] \text{ of}
\]

\[
\begin{align*}
| [\Gamma \vdash \#p \ldots] \Rightarrow & \text{lookup } [\Gamma] [\Gamma \vdash \#p \ldots] \; \text{rs} & \% \text{ Variable} \\
| [\Gamma \vdash \text{app} \; (M1 \ldots) \; (M2 \ldots)] \Rightarrow & \text{let } \text{Arr } \text{ha } f = \text{main } [\Gamma] [\Gamma \vdash M1 \ldots] \; \text{rs } \text{in} \\
& f [\vdash _{\ldots}] (\text{main } [\Gamma] [\Gamma \vdash M2 \ldots] \; \text{rs}) & \% \text{ Application} \\
| [\Gamma \vdash \text{lam} \; (\lambda x. \; M1 \ldots \; x)] \Rightarrow & \text{Arr } [\vdash \text{h/value} \; s/\text{refl} \; v/\text{lam}] \\
& (\text{mlam } N \Rightarrow \text{fn } rN \Rightarrow \text{closed } [\vdash \text{s/beta}] \\
& \quad (\text{main } [\Gamma,x : \text{tm} \; _{\ldots}] [\Gamma,x \vdash M1 \ldots \; x] \; (\text{Cons } \text{rs } rN))) & \% \text{ Abstraction} \\
| [\Gamma \vdash c] \Rightarrow & \text{I } [\vdash \text{h/value} \; s/\text{refl} \; v/c]; & \% \text{ Constant}
\end{align*}
\]
**Fundamental Lemma**

```plaintext
rec closed : [ ⊢ mstep M M'] → Reduce [ ⊢ A] [ ⊢ M'] →Reduce [ ⊢ A] [ ⊢ M] = ? ;
rec main : {Γ:ctx}{M:[Γ ⊢ tm A]} RedSub [ ⊢ σ] →Reduce [ ⊢ A] [ ⊢ M σ] =
mlam Γ ⇒ mlam M ⇒ fn rs ⇒ case [Γ ⊢ M ...] of
| [Γ ⊢ #p ...] ⇒ lookup [Γ] [Γ ⊢ #p ...] rs % Variable
| [Γ ⊢ app (M1 ...) (M2 ...)] ⇒
  let Arr ha f = main [Γ] [Γ ⊢ M1 ...] rs in
  f [ ⊢ _ ] (main [Γ] [Γ ⊢ M2 ...] rs)
| [Γ ⊢ lam (λx. M1 ... x)] ⇒
  Arr [ ⊢ h/value s/refl v/lam]
  (mlam N ⇒ fn rN ⇒ closed [ ⊢ s/beta]
  (main [Γ,x:tm _] [Γ,x ⊢ M1 ... x] (Cons rs rN)))
| [Γ ⊢ c] ⇒ I [ ⊢ h/value s/refl v/c]; % Constant
```

- Direct encoding of on-paper proof
- Equations about substitution properties automatically discharged (amounts to roughly a dozen lemmas about substitution and weakening)
- Total encoding about 75 lines of Beluga code
This Talk

Design and implementation of Beluga

- Introduction
- Example: Proof by logical relations
- Writing a proof in Beluga ...
- Conclusion and current work
Revisiting the Design of Beluga

- **Level 1: Contextual LF**

<table>
<thead>
<tr>
<th>On paper proof</th>
<th>In Beluga [IJCAR’10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well-formed derivations</td>
<td>Dependent types</td>
</tr>
<tr>
<td>Renaming, Substitution</td>
<td>$\alpha$-renaming, $\beta$-reduction in LF</td>
</tr>
</tbody>
</table>
## Revisiting the Design of Beluga

### Level 1: Contextual LF

<table>
<thead>
<tr>
<th>On paper proof</th>
<th>In Beluga [IJCAR’10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well-formed derivations</td>
<td>Dependent types</td>
</tr>
<tr>
<td>Renaming, Substitution</td>
<td>$\alpha$-renaming, $\beta$-reduction in LF</td>
</tr>
<tr>
<td>Well-scoped derivation</td>
<td>Contextual types and objects [TOCL’08]</td>
</tr>
<tr>
<td>Context</td>
<td>Context schemas</td>
</tr>
<tr>
<td>Properties of contexts (weakening,</td>
<td>Typing for schemas</td>
</tr>
<tr>
<td>uniqueness)</td>
<td></td>
</tr>
<tr>
<td>Substitutions (composition, identity)</td>
<td>Substitution type [LFMTP’13]</td>
</tr>
</tbody>
</table>
## Revisiting the Design of Beluga

- **Level 1: Contextual LF**

<table>
<thead>
<tr>
<th>On paper proof</th>
<th>In Beluga [IJCAR’10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well-formed derivations</td>
<td>Dependent types</td>
</tr>
<tr>
<td>Renaming, Substitution</td>
<td>$\alpha$-renaming, $\beta$-reduction in LF</td>
</tr>
<tr>
<td>Well-scoped derivation</td>
<td>Contextual types and objects [TOCL’08]</td>
</tr>
<tr>
<td>Context</td>
<td>Context schemas</td>
</tr>
<tr>
<td>Properties of contexts</td>
<td>Typing for schemas</td>
</tr>
<tr>
<td>(weakening, uniqueness)</td>
<td></td>
</tr>
<tr>
<td>Substitutions</td>
<td>Substitution type [LFMTP’13]</td>
</tr>
<tr>
<td>(composition, identity)</td>
<td></td>
</tr>
</tbody>
</table>

- **Level 2: Functional programming with indexed types [POPL’08, POPL’12]**

  | Case analysis | Case analysis and pattern matching |
  | Inversion | Pattern matching using let-expression |
  | Induction hypothesis | Recursive call |
Other Examples and Comparison

- Other examples using logical relations:
  - Weak normalization which evaluates under lambda-abstraction
  - Algorithmic equality for LF (A. Cave) (draft available)

  Sufficient evidence that Beluga is ideally suited to support such advanced proofs

- Comparison (concentrating on the given weak normalization proof)
  - Coq/Agda formalization with well-scoped de Bruijn indices: dozen additional lemmas
  - Abella: 4 additional lemmas and diverges a bit from on-paper proof
  - Twelf: Too weak to for directly encoding such proofs; Implement auxiliary logic.
What Have We Achieved?

- Foundation for programming proofs in context [POPL’12]
  - Proof term language for first-order logic over contextual LF as domain
  - Uniform treatment of contextual types, context, ... 
  - Modular foundation for dependently-typed programming with phase-distinction ⇒ Generalization of DML and ATS

- Extending contextual LF with first-class substitutions and their equational theory [LFMTP’13]

- Rich set of examples
  - Type-preserving compiler for simply typed lambda-calculus (joint work with O. Savary Belanger, S. Monnier [CPP’13])
  - (Weak) Normalization proofs (A. Cave)

- Latest release in Jan’14: Support for indexed data types, first-class substitutions, equational theory behind substitutions

“A language that doesn’t affect the way you think about programming, is not worth knowing.” - Alan Perlis
Current Work

- Prototype in OCaml (ongoing - next release Aug 2014) providing an interactive programming mode
- Structural recursion (S. S. Ruan, A. Abel)
  Develops a foundation of structural recursive functions for Beluga; proof of normalization; prototype implementation under way
- Coinduction in Beluga (D. Thibodeau, A. Cave)
  Extending work on simply-typed copatterns [POPL’13] to Beluga
- Case study:
  - Certified compiler (O. Savary Belanger, CPP’13)
  - Proof-carrying authorization with constraints (Tao Xue)
- Extending Beluga to full dependent types (A. Cave)
- Type reconstruction for dependently typed programs (F. Ferreira, PPDP’14)
- ORBI - Benchmarks for comparing systems supporting HOAS encodings (A. Felty, A. Momigliano, March 2014)
Thank you!

Download prototype and examples at

http://complogic.cs.mcgill.ca/beluga/

Current Belugians: Brigitte Pientka, Mathias Puech, Tao Xue, Olivier Savary Belanger, Andrew Cave, Francisco Ferreira, Stefan Monnier, David Thibodeau, Sherry Shanshan Ruan, Shawn Otis