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 curent 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 curent work

"The limits of my language mean the limits of my world."
- L. Wittgenstein

Simply Typed Lambda-calculus (Gentzen-style)

Types and Terms

Types $A, B ::= \mathsf{i} \mid A \to B$

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

Evaluation Judgment: $\boxed{M \longrightarrow M'}$ read as "M steps to M'"

$$\frac{}{\mathsf{app}\ (\mathsf{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathsf{s/beta} \qquad \frac{}{M \longrightarrow M}\ \mathsf{s/refl}$$

$$\frac{M \longrightarrow M'}{\mathsf{app}\ M\ N \longrightarrow \mathsf{app}\ M'\ N}\ \mathsf{s/app} \qquad \frac{M \longrightarrow M' \quad M' \longrightarrow N}{M \longrightarrow N}\ \mathsf{s/trans}$$

Simply Typed Lambda-calculus (Gentzen-style)

Types and Terms

Types $A, B ::= \mathsf{i} \mid A \to B$

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

Evaluation Judgment: $\boxed{M \longrightarrow M'}$ read as "M steps to M'"

$$\frac{}{\mathsf{app}\ (\mathsf{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathsf{s/beta} \qquad \frac{}{M \longrightarrow M}\ \mathsf{s/refl}$$

$$\frac{M \longrightarrow M'}{\mathsf{app}\ M\ N \longrightarrow \mathsf{app}\ M'\ N}\ \mathsf{s/app} \qquad \frac{M \longrightarrow M' \quad M' \longrightarrow N}{M \longrightarrow N}\ \mathsf{s/trans}$$

Typing Judgment: $\boxed{M : A}$ read as "M has type A" (Gentzen-style)

$$\frac{}{\mathbf{c} : \mathbf{i}}\ \mathsf{const} \qquad \frac{\begin{array}{c}\overline{x : A}\ u \\ \vdots \\ M : B\end{array}}{(\mathsf{lam}\ x.M) : (A \to B)}\ \mathsf{lam}^{x,u} \qquad \frac{M : (A \to B) \quad N : A}{(\mathsf{app}\ M\ N) : B}\ \mathsf{app}$$

Simply Typed Lambda-calculus with Contexts

Types and Terms

Types $A, B ::= \mathsf{i} \mid A \rightarrow B$

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

Evaluation Judgment: $\boxed{M \longrightarrow M'}$ read as "M steps to M'"

$$\frac{}{\mathsf{app}\ (\mathsf{lam} x.M)\ N \longrightarrow [N/x]M}\ \mathsf{s/beta} \qquad \frac{}{M \longrightarrow M}\ \mathsf{s/refl}$$

$$\frac{M \longrightarrow M'}{\mathsf{app}\ M\ N \longrightarrow \mathsf{app}\ M'\ N}\ \mathsf{s/app} \qquad \frac{M \longrightarrow M' \quad M' \longrightarrow N}{M \longrightarrow N}\ \mathsf{s/trans}$$

Typing Judgment: $\boxed{\Gamma \vdash M : A}$ read as "M has type A in context Γ"

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

Context $\Gamma ::= \cdot \mid \Gamma, x : A$ We are introducing the variable x together with the assumption $x : A$

Talking about Derivations

Typing rules

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\mathsf{lam}\, x.M) : (A \to B)}\ \mathsf{lam}^x \qquad \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\mathsf{app}\ M\ N) : B}\ \mathsf{app}$$

Evaluation rules

$$\frac{}{\mathsf{app}\ (\mathsf{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathsf{beta} \qquad \frac{M \longrightarrow M'}{\mathsf{app}\ M\ N \longrightarrow \mathsf{app}\ M'\ N}\ \mathsf{app}$$

Talking about Derivations

Typing rules

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\mathsf{lam}\, x.M) : (A \to B)}\ \mathsf{lam}^x \qquad \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\mathsf{app}\ M\ N) : B}\ \mathsf{app}$$

Evaluation rules

$$\frac{}{\mathsf{app}\ (\mathsf{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathsf{beta} \qquad \frac{M \longrightarrow M'}{\mathsf{app}\ M\ N \longrightarrow \mathsf{app}\ M'\ N}\ \mathsf{app}$$

- What kinds of variables are used?

Talking about Derivations

Typing rules

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\mathrm{lam}\, x.M) : (A \to B)}\ \mathrm{lam}^x \qquad \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\mathrm{app}\ M\ N) : B}\ \mathrm{app}$$

Evaluation rules

$$\frac{}{\mathrm{app}\ (\mathrm{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathrm{beta} \qquad \frac{M \longrightarrow M'}{\mathrm{app}\ M\ N \longrightarrow \mathrm{app}\ M'\ N}\ \mathrm{app}$$

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

Talking about Derivations

Typing rules

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\mathsf{lam}\, x.M) : (A \to B)}\ \mathsf{lam}^x \qquad \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\mathsf{app}\ M\ N) : B}\ \mathsf{app}$$

Evaluation rules

$$\frac{}{\mathsf{app}\ (\mathsf{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathsf{beta} \qquad \frac{M \longrightarrow M'}{\mathsf{app}\ M\ N \longrightarrow \mathsf{app}\ M'\ N}\ \mathsf{app}$$

- 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

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\mathrm{lam}\, x.M) : (A \to B)}\ \mathrm{lam}^x \qquad \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\mathrm{app}\ M\ N) : B}\ \mathrm{app}$$

Evaluation rules

$$\frac{}{\mathrm{app}\ (\mathrm{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathrm{beta} \qquad \frac{M \longrightarrow M'}{\mathrm{app}\ M\ N \longrightarrow \mathrm{app}\ M'\ N}\ \mathrm{app}$$

- 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

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\mathrm{lam}\, x.M) : (A \to B)}\ \mathrm{lam}^x \qquad \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\mathrm{app}\ M\ N) : B}\ \mathrm{app}$$

Evaluation rules

$$\frac{}{\mathrm{app}\ (\mathrm{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathrm{beta} \qquad \frac{M \longrightarrow M'}{\mathrm{app}\ M\ N \longrightarrow \mathrm{app}\ M'\ N}\ \mathrm{app}$$

- 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

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\mathrm{lam}\, x.M) : (A \to B)}\ \mathrm{lam}^x \qquad \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\mathrm{app}\ M\ N) : B}\ \mathrm{app}$$

Evaluation rules

$$\frac{}{\mathrm{app}\ (\mathrm{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathrm{beta} \qquad \frac{M \longrightarrow M'}{\mathrm{app}\ M\ N \longrightarrow \mathrm{app}\ M'\ N}\ \mathrm{app}$$

- 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

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\mathrm{lam}\, x.M) : (A \to B)}\ \mathrm{lam}^x \qquad \frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : B}{\Gamma \vdash (\mathrm{app}\ M\ N) : B}\ \mathrm{app}$$

Evaluation rules

$$\frac{}{\mathrm{app}\ (\mathrm{lam}\ x.M)\ N \longrightarrow [N/x]M}\ \mathrm{beta} \qquad \frac{M \longrightarrow M'}{\mathrm{app}\ M\ N \longrightarrow \mathrm{app}\ M'\ N}\ \mathrm{app}$$

- 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

Weak Normalization for Simply Typed Lambda-calculus

Theorem

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

Weak Normalization for Simply Typed Lambda-calculus

Theorem

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

Proof.

1 Define reducibility candidate $\mathcal{R}_A$

$$\begin{array}{lcl} \mathcal{R}_{\mathbf{i}} & = & \{M \mid M \text{ halts}\} \\ \mathcal{R}_{A \to B} & = & \{M \mid M \text{ halts and } \forall N \in \mathcal{R}_A, (\text{app } M\ N) \in \mathcal{R}_B\} \end{array}$$

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

3 Backwards closed: If $M' \in \mathcal{R}_A$ and $M \longrightarrow M'$ 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 $\mathcal{D} : \Gamma \vdash M : A$ *and* $\sigma \in \mathcal{R}_\Gamma$ *then* $[\sigma]M \in \mathcal{R}_A$.

where $\sigma \in \mathcal{R}_\Gamma$ is defined as:

$$\frac{}{\cdot \in \mathcal{R}_\cdot} \qquad \frac{\sigma \in \mathcal{R}_\Gamma \quad N \in \mathcal{R}_A}{(\sigma, N/x) \in \mathcal{R}_{\Gamma, x:A}}$$

Generalization of Fundamental Lemma

Lemma (Main lemma)

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

Generalization of Fundamental Lemma

Lemma (Main lemma)

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

Proof.

Case $\mathcal{D} = \dfrac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \mathit{var}$

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

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

Generalization of Fundamental Lemma

Lemma (Main lemma)

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

Proof.

Case $\mathcal{D} = \dfrac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \mathit{var}$

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

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

Case $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma \vdash M : A \to B} \qquad \overset{\mathcal{D}_2}{\Gamma \vdash N : A}}{\Gamma \vdash \text{app } M\ N : B}\ \mathit{app}$

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

$N \in \mathcal{R}_A$ by i.h. $\mathcal{D}_2$

$M \in \mathcal{R}_{A \to B}$ by i.h. $\mathcal{D}_1$

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

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

Generalization of Fundamental Lemma

Lemma (Main lemma)

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

Proof.

Case $\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Gamma, x{:}A \vdash M : B\end{matrix}}{\Gamma \vdash \text{lam } x.M : A \to B}\ \mathit{lam}$

$[\sigma](\text{lam } x.M) = \text{lam } x.([\sigma, x/x]M)$ by properties of substitution

`halts` $(\text{lam } x.[\sigma, x/x]M)$ since it is a value

Suppose $N \in \mathcal{R}_A$.

$[\sigma, N/x]M \in \mathcal{R}_B$ by I.H. on $\mathcal{D}_1$ since $\sigma \in \mathcal{R}_\Gamma$

$[N/x][\sigma, x/x]M \in \mathcal{R}_B$ by properties of substitution

$\text{app } (\text{lam } x.\ [\sigma, x/x]M)\ N \in \mathcal{R}_B$ by Backwards closure

Hence $[\sigma](\text{lam } x.M) \in \mathcal{R}_{A \to B}$ by definition □

Challenging Benchmark

- Model different level of bindings
  lambda-binder, $\forall$ in reducibility definition $\mathcal{R}$, quantification over substitutions and contexts
- Simultanous substitution and algebraic properties
  Substitution lemma, Reason about composition, decomposition, associativity, identity, etc.

  $$\begin{array}{lcl} [\cdot]M & = & M \\ [\sigma, N/x]M & = & [N/x][\sigma, x/x]M \\ [\sigma_1][\sigma_2]M & = & [[\sigma_1]\sigma_2]M \end{array}$$

  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

This Talk

Design and implementation of Beluga

- Introduction
- Example: Proof by logical relations
- Writing a proof in Beluga . . .
- Conclusion and curent 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 α-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
  $\leadsto$ support for α-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 specifc 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
  $\rightsquigarrow$ support for α-renaming, substitution, adequate representations
- Contextual LF: Contextual types characterize contextual objects [TOCL'08]
  $\rightsquigarrow$ support well-scoped derivations
  $\rightsquigarrow$ 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 specifc domain ($=$ contextual LF) together with domain-specific induction principle and recursive definitions ($=$ indexed recursive types)

On paper proof	Proofs as functions in Beluga
Case analysis	Case analysis and pattern matching
Inversion	Pattern matching using let-expression
Induction Hypothesis	Recursive call

Step 1: Represent Types and Lambda-terms in LF

Types and Terms

$$\text{Types } A, B ::= \text{i} \mid A \to B \qquad \text{Terms M, N} ::= x \mid \mathbf{c} \mid \text{lam } x.M \mid \text{app } M\ N$$

Typing rules

$$\frac{}{\mathbf{c} : \mathbf{i}}\ \text{const} \qquad \frac{\begin{array}{c}\overline{x : A}^{\,u} \\ \vdots \\ M : B\end{array}}{(\text{lam}\, x.M) : (A \to B)}\ \text{lam}^x \qquad \frac{M : (A \to B) \quad N : A}{(\text{app } M\ N) : B}\ \text{app}$$

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;
```

Step 1: Represent Types and Lambda-terms in LF

Types and Terms

Types $A, B ::= \mathrm{i} \mid A \to B$

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

Typing rules

$$\frac{}{\mathbf{c} : \mathbf{i}}\ \text{const} \qquad \frac{\begin{array}{c}\overline{x : A}^{\,u} \\ \vdots \\ M : B\end{array}}{(\mathrm{lam}\, x.M) : (A \to B)}\ \mathrm{lam}^{x} \qquad \frac{M : (A \to B) \quad N : A}{(\mathrm{app}\ M\ N) : B}\ \text{app}$$

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 $M \in \mathcal{R}_A$:

$$\begin{array}{lcl} \mathcal{R}_{\mathbf{i}} & = & \{M \mid \texttt{halts}\ M\} \\ \mathcal{R}_{A \to B} & = & \{M \mid \texttt{halts}\ M \text{ and } \forall N \in \mathcal{R}_A, (\mathsf{app}\ M\ N) \in \mathcal{R}_B\} \end{array}$$

Reducibility Candidates as Indexed Types

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

$$\begin{array}{lcl} \mathcal{R}_{\mathbf{i}} & = & \{M \mid \texttt{halts}\ M\} \\ \mathcal{R}_{A\to B} & = & \{M \mid \texttt{halts}\ M \text{ and } \forall N \in \mathcal{R}_A, (\mathsf{app}\ M\ N) \in \mathcal{R}_B\} \end{array}$$

Computation-level data types in Beluga

```
datatype Reduce : {A:[ ⊢ tp]} {M:[ ⊢ tm A]} ctype =
| I   : [ ⊢ halts M] → Reduce [ ⊢ i] [⊢ M]
| Arr : [ ⊢ halts M] →
        ({N:[ ⊢ tm A]} Reduce [ ⊢ A] [ ⊢ N] → Reduce [ ⊢ B] [ ⊢ app M N])
    → Reduce [ ⊢ arr A B] [ ⊢ M];
```

- `[⊢ app M N]` and `[ ⊢ arr A B]` are contextual types [TOCL'08].
- Note: → is overloaded.
  - → is the LF function space : binders in the object language are modelled by LF functions (used inside `[ ]`)
  - → is a computation-level function (used outside `[ ]`)
- Not strictly positive definition, but stratified.

Reducibility Candidates as Indexed Types

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

$$\frac{}{\cdot \in \mathcal{R}_{\cdot}} \qquad \frac{\sigma \in \mathcal{R}_\Gamma \quad N \in \mathcal{R}_A}{(\sigma, N/x) \in \mathcal{R}_{\Gamma, x:A}}$$

Reducibility Candidates as Indexed Types

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

$$\frac{}{\cdot \in \mathcal{R}_\cdot} \qquad \frac{\sigma \in \mathcal{R}_\Gamma \quad N \in \mathcal{R}_A}{(\sigma, N/x) \in \mathcal{R}_{\Gamma, x:A}}$$

Computation-level data types in Beluga

```
datatype RedSub : (Γ:ctx){σ: ⊢ Γ} ctype =
| Nil : RedSub [ ⊢ ^ ]
| Cons : RedSub [ ⊢ σ] → Reduce [ ⊢ A] [ ⊢ M] → RedSub [ ⊢ σ M ];
```

- Contexts are structured sequences and are classified by context schemas

  schema `ctx = x:tm A.`

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

Theorems as Computation-level Types

Lemma (Backward closed)

If $M \longrightarrow M'$ and $M' \in \mathcal{R}_A$ then $M \in \mathcal{R}_A$.

```
rec closed : [⊢ mstep M M’] → Reduce [⊢ A] [⊢ M’] → Reduce [⊢ A] [⊢ M] = ? ;
```

Lemma (Main lemma)

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

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

Fundamental Lemma

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] =
```

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
```

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

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

| [Γ ⊢ lam (λx. M1 ... x)] ⇒                                        % Abstraction
  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)))
```

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

| [Γ ⊢ lam (λx. M1 ... x)] ⇒                                  % Abstraction
  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
```

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

| [Γ ⊢ lam (λx. M1 ... x)] ⇒                                 % Abstraction
  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 curent work

Revisiting the Design of Beluga

- Level 1: Contextual LF

On paper proof	In Beluga [IJCAR'10]
Well-formed derivations	Dependent types
Renaming,Substitution	α-renaming, β-reduction in LF

Revisiting the Design of Beluga

- Level 1: Contextual LF

On paper proof	In Beluga [IJCAR'10]
Well-formed derivations	Dependent types
Renaming,Substitution	α-renaming, β-reduction in LF
Well-scoped derivation	Contextual types and objects [TOCL'08]
Context	Context schemas
Properties of contexts (weakening, uniqueness)	Typing for schemas
Substitutions (composition, identity)	Substitution type [LFMTP'13]

Revisiting the Design of Beluga

- Level 1: Contextual LF

On paper proof	In Beluga [IJCAR'10]
Well-formed derivations	Dependent types
Renaming,Substitution	α-renaming, β-reduction in LF
Well-scoped derivation	Contextual types and objects [TOCL'08]
Context	Context schemas
Properties of contexts (weakening, uniqueness)	Typing for schemas
Substitutions (composition, identity)	Substitution type [LFMTP'13]

- 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)

  $\Longrightarrow$ 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)

The End

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