Mechanizing Meta-Theory in Beluga

Brigitte Pientka

School of Computer Science
McGill University
Montreal, Canada

Joint work with Andrew Cave

How to mechanize 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 (see: certified code, proof-carrying architectures).

How to mechanize 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 (see: certified code, proof-carrying architectures).

Program
(in Assembler, C, ML, Java, ...)

Meta-Theory

Properties

– Memory Safety:
Program does not crash

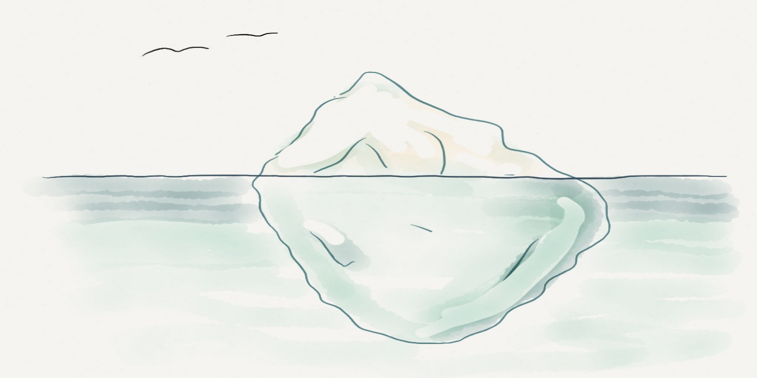
– Authenticity:
Communicates only within domain mcgill.ca

– Type Safety:
Execution of program does not go wrong

Question

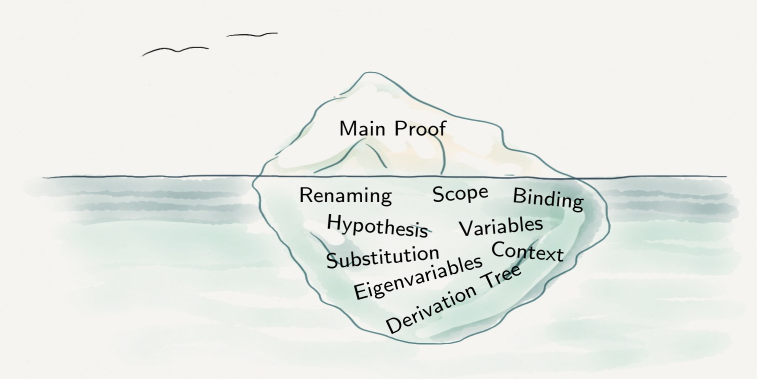
What are good meta-languages to program and reason with formal systems and proofs?

Proofs: The tip of the iceberg



"We may think of [the] proof as an iceberg. In the top of it, we find what we usually consider the real proof; underwater, the most of the matter, consisting of all mathematical preliminaries a reader must know in order to understand what is going on." *S. Berardi [1990]*

Proofs: The tip of the iceberg



"We may think of [the] proof as an iceberg. In the top of it, we find what we usually consider the real proof; underwater, the most of the matter, consisting of all mathematical preliminaries a reader must know in order to understand what is going on." *S. Berardi [1990]*

BELUGA: Programming Proofs in Context

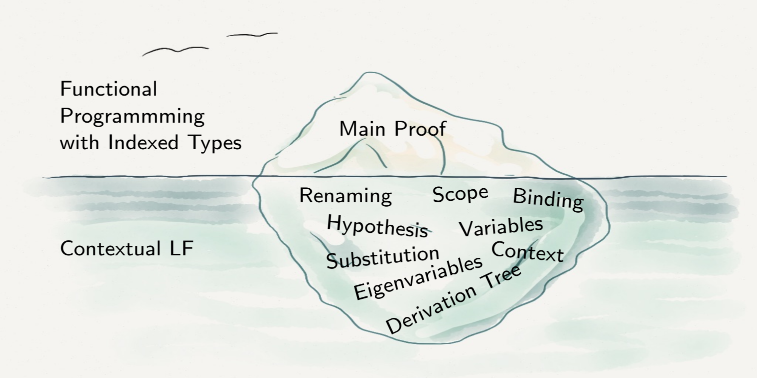
"The motivation behind the work in very-high-level languages is to ease the programming task by providing the programmer with a language containing primitives or abstractions suitable to his problem area. The programmer is then able to spend his effort in the right place; he concentrates on solving his problem, and the resulting program will be more reliable as a result. Clearly, this is a worthwhile goal." B. Liskov [1974]

BELUGA: Programming Proofs in Context

"The motivation behind the work in very-high-level languages is to ease the programming task by providing the programmer with a language containing primitives or abstractions suitable to his problem area. The programmer is then able to spend his effort in the right place; he concentrates on solving his problem, and the resulting program will be more reliable as a result. Clearly, this is a worthwhile goal." B. Liskov [1974]

Above and Below the Surface

Beluga: Dependently typed Programming and Proof Environment



- Below the surface: Support for key concepts based on Contextual LF
- Above the surface: Proofs by structural Induction = Recursive Programs
  First-order Logic over Contextual LF objects (i.e. Contexts, Derivation trees, Substitutions, ...) together with inductive definitions and induction principles

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

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'"

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

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

Simply Typed Lambda-calculus (Gentzen-style)

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

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

Simply Typed Lambda-calculus with Contexts

Types and Terms

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

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

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

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

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

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

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

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

Derivations Under the Magnifying Glass

Typing rules

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

Evaluation rules

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

Derivations Under the Magnifying Glass

Typing rules

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

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?

Derivations Under the Magnifying Glass

Typing rules

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

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, Eigenvariables, Schematic variables, Context variables

Derivations Under the Magnifying Glass

Typing rules

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

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, Eigenvariables, Schematic variables, Context variables
- What operations on variables are needed?

Derivations Under the Magnifying Glass

Typing rules

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

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, Eigenvariables, Schematic variables, Context variables
- What operations on variables are needed? Substitution and Renaming for bound variable, Substitution for schematic variables, Substitution for hypothesis and eigenvariables

Derivations Under the Magnifying Glass

Typing rules

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

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, Eigenvariables, Schematic variables, Context variables
- What operations on variables are needed? Substitution and Renaming for bound variable, Substitution for schematic variables, Substitution for hypothesis and eigenvariables
- How should we represent contexts? What properties do contexts have?

Derivations Under the Magnifying Glass

Typing rules

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \text{lam}\, x.M : A \Rightarrow B}\ \text{lam}^x \qquad \frac{\Gamma \vdash M : A \Rightarrow B \quad \Gamma \vdash N : B}{\Gamma \vdash \text{app}\ M\ N : B}\ \text{app}$$

Evaluation rules

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

- What kinds of variables are used? Bound variables, Eigenvariables, Schematic variables, Context variables
- What operations on variables are needed? Substitution and Renaming for bound variable, Substitution for schematic variables, Substitution for hypothesis and eigenvariables
- How should we represent contexts? What properties do contexts have? (Structured) sequences, Uniqueness of declaration, Weakening, Substitution lemma, etc.

Derivations Under the Magnifying Glass

Typing rules

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

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, Eigenvariables, Schematic variables, Context variables
- What operations on variables are needed? Substitution and Renaming for bound variable, Substitution for schematic variables, Substitution for hypothesis and eigenvariables
- 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.

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 $\mathcal{R}_A$

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

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:

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

Generalization of Fundamental Lemma

Lemma (Main lemma)

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

Proof.

Case $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ \Gamma, x{:}A \vdash M : B\end{array}}{\Gamma \vdash \mathrm{lam}\ x.M : A \Rightarrow B}\ lam$

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

`halts` $(\mathrm{lam}\ x.[\sigma, x/x]M)$ 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

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

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

$\square$

Challenging Benchmark

"I discovered that the core part of the proof (here proving lemmas about CR) is fairly straightforward and only requires a good understanding of the paper version. However, in completing the proof I observed that in certain places I had to invest much more work than expected, e.g. proving lemmas about substitution and weakening." *T. Altenkirch [TLCA'93]*

Challenging Benchmark

"I discovered that the core part of the proof (here proving lemmas about CR) is fairly straightforward and only requires a good understanding of the paper version. However, in completing the proof I observed that in certain places I had to invest much more work than expected, e.g. proving lemmas about substitution and weakening." *T. Altenkirch [TLCA'93]*

- Binders: lambda-binder, $\forall$ in reducibility definition, quantification over substitutions and contexts
- Contexts: Uniqueness of assumptions, weakening, etc.
- Simultanous substitution and algebraic properties:
  Substitution lemma, 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

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

The Top: Functional programming with indexed types [POPL'08,POPL'12]

Proof term language for first-order logic over a specifc domain (= contextual LF) together inductive definitions (= relations) about domain objects and domain-specific induction principle [TLCA'15]

Beluga$^{\mu}$: Two Level Approach

The Top: Functional programming with indexed types [POPL'08,POPL'12]

Proof term language for first-order logic over a specifc domain (= contextual LF) together inductive definitions (= relations) about domain objects and domain-specific induction principle [TLCA'15]

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

The Bottom: Contextual logical framework LF [HHP'93,TOCL'08]

- Compact representation of formal systems and derivations
- Higher-order abstract syntax trees and dependent types
  $\rightsquigarrow$ support for α-renaming, substitution, adequate representations

Beluga$^{\mu}$: Two Level Approach

The Top: Functional programming with indexed types [POPL'08,POPL'12]

Proof term language for first-order logic over a specifc domain (= contextual LF) together inductive definitions (= relations) about domain objects and domain-specific induction principle [TLCA'15]

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

The Bottom: Contextual logical framework LF [HHP'93,TOCL'08]

- Compact representation of formal systems and derivations
- Higher-order abstract syntax trees 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]

Beluga$^{\mu}$: Two Level Approach

The Top: Functional programming with indexed types [POPL'08,POPL'12]

Proof term language for first-order logic over a specifc domain (= contextual LF) together inductive definitions (= relations) about domain objects and domain-specific induction principle [TLCA'15]

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

The Bottom: Contextual logical framework LF [HHP'93,TOCL'08]

- Compact representation of formal systems and derivations
- Higher-order abstract syntax trees 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]

Step 1: Represent Types and Lambda-terms in LF

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

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

Typing rules

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

LF representation in Beluga

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

```
LF 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 $A, B ::= \mathsf{i} \mid A \Rightarrow B$

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

Typing rules

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

LF representation in Beluga

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

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

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

Reducibility Candidates as Indexed Types

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

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

Computation-level data types in Beluga

```
stratified Reduce : {A:[ ⊢ tp]} {M:[ ⊢ tm A]} type =
| 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 $\sigma \in \mathcal{R}_\Gamma$:

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

Computation-level data types in Beluga

```
inductive  RedSub : (Γ:ctx){σ: ⊢ Γ} type =
| 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] ⇒                                        % Abstraction
  Arr [ ⊢ h/value s/refl v/lam]
   (mlam N ⇒ fn rN ⇒ closed [ ⊢ s/beta]
                              (main [Γ,x:tm _] [Γ,x ⊢ M1] (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] ⇒                                        % Abstraction
  Arr [ ⊢ h/value s/refl v/lam]
   (mlam N ⇒ fn rN ⇒ closed [ ⊢ s/beta]
                          (main [Γ,x:tm _] [Γ,x ⊢ M1] (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] ⇒                                         % Abstraction
  Arr [ ⊢ h/value s/refl v/lam]
   (mlam N ⇒ fn rN ⇒ closed [ ⊢ s/beta]
                          (main [Γ,x:tm _] [Γ,x ⊢ M1] (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

- Top : 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

- Bottom: Contextual LF

On paper proof	In Beluga [IJCAR'10,CADE'15]
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]

Alternatives

General Theorem Proving Environments

- Calculus of Construction (Coq) / Martin Löf Type Theory (Agda)
  No special support for variables, assumptions, derivation trees, etc.
  About a dozen extra lemmas
- Isabelle / Nominal
  support for variable names, but not for assumptions, derivation trees, etc.
  based on nominal set theory; about a dozen extra lemmas

Alternatives

General Theorem Proving Environments

- Calculus of Construction (Coq) / Martin Löf Type Theory (Agda)
  No special support for variables, assumptions, derivation trees, etc.
  About a dozen extra lemmas
- Isabelle / Nominal
  support for variable names, but not for assumptions, derivation trees, etc.
  based on nominal set theory; about a dozen extra lemmas

Domain-specific Provers (Higher-Order Abstract Syntax (HOAS))

- Abella: encode second-order hereditary Harrop (HH) logic in $\mathcal{G}$, an extension of first-order logic with a new quantifier ∇, and develop inductive proofs in $\mathcal{G}$ by reasoning about the size of HH derivations .
  diverges a bit from on-paper proof; 4 additional lemmas
- Twelf: Too weak for directly encoding such proofs; implement auxiliary logic.

Current Work

- Prototype in OCaml (ongoing - last release March 2015)
  providing an interactive programming mode, totality checker [CADE'15]

  https://github.com/Beluga-lang/Beluga

- Mechanizing Types and Programming Languages - A companion:

  https://github.com/Beluga-lang/Meta

- Coinduction in Beluga (D. Thibodeau, A. Cave)
  Extending work on simply-typed copatterns [POPL'13] to Beluga
  Long term: reason about reactive systems [POPL'14]
- Case study: Certified compiler (O. Savary Belanger) [CPP'13]
- Extending Beluga to full dependent types (A. Cave)
- Type reconstruction (F. Ferreira [PPDP'14] and [JFP'13])
- ORBI - Benchmarks for comparing systems supporting HOAS encodings [JAR'15,LFMTP'15] (A. Felty, A. Momigliano, March 2015)

The End

Thank you!

Download prototype and examples at

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

Thanks go to: Andrew Cave, Joshua Dunfield, Olivier Savary Belanger, Matthias Boespflug, Scott Cooper, Francisco Ferreira, Aidan Marchildon, Stefan Monnier, Agata Murawska, Nicolas Jeannerod, David Thibodeau, Shawn Otis, Rohan Jacob Rao, Shanshan Ruan, Tao Xue

"A language that doesn't affect the way you think about programming, is not worth knowing." - Alan Perlis