POPLMark Reloaded: Mechanizing Logical Relations Proofs

Brigitte Pientka

McGill University

Joint work with A. Abel (Chalmers), A. Hameer (McGill), A. Momigliano (Milan), S. Schäfer (Saarbrücken), K. Stark (Saarbrücken)

Mechanizing formal systems together with proofs establishes trust.

Mechanizing formal systems together with proofs establishes trust. . . and finds flaws.

Programs go wrong.

Programs go wrong.

Testing correctness of C Compilers [Le et.al PLDI'14]:

- GCC and LLVM had over 195 bugs
- Compcert the only compiler where no bugs were found

Programming lang. designs and implementations go wrong.

Type Safety of Java (20 years ago)

Java is Type Safe — Probably

Sophia Drossopoulou and Susan Eisenbach

Department of Computing
Imperial College of Science, Technology and Medicine
email: sd and se @doc.ic.ac.uk

Abstract. Amidst rocketing numbers of enthusiastic Java programmers and internet applet users, there is growing concern about the security of executing Java code produced by external, unknown sources. Rather than waiting to find out empirically what damage Java programs do, we aim to examine first the language and then the environment looking for points of weakness. A proof of the soundness of the Java type system is a first, necessary step towards demonstrating which Java programs won't compromise computer security.

We consider a type safe subset of Java describing primitive types, classes, inheritance, instance variables and methods, interfaces, shadowing, dynamic method binding, object creation, null and arrays. We argue that for this subset the type system is sound, by proving that program execution preserves the types, up to subclasses/subinterfaces.

Programming lang. designs and implementations go wrong.

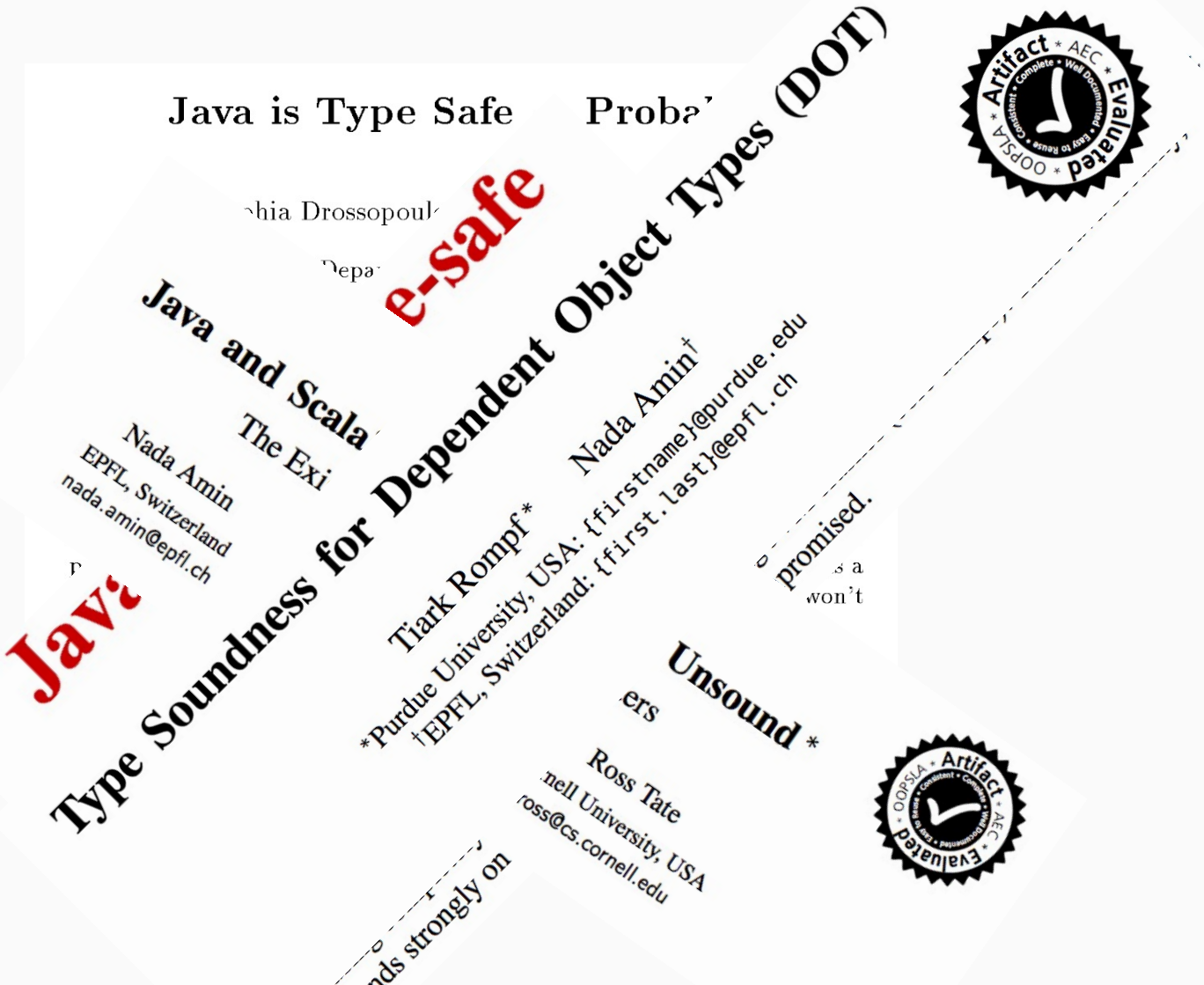
Type Safety of Java (20 years ago)

Programming lang. designs and implementations go wrong.

Type Safety of Java and Scala (20 years later)

Programming lang. designs and implementations go wrong.

Type Safety of Java and Scala (20 years later)

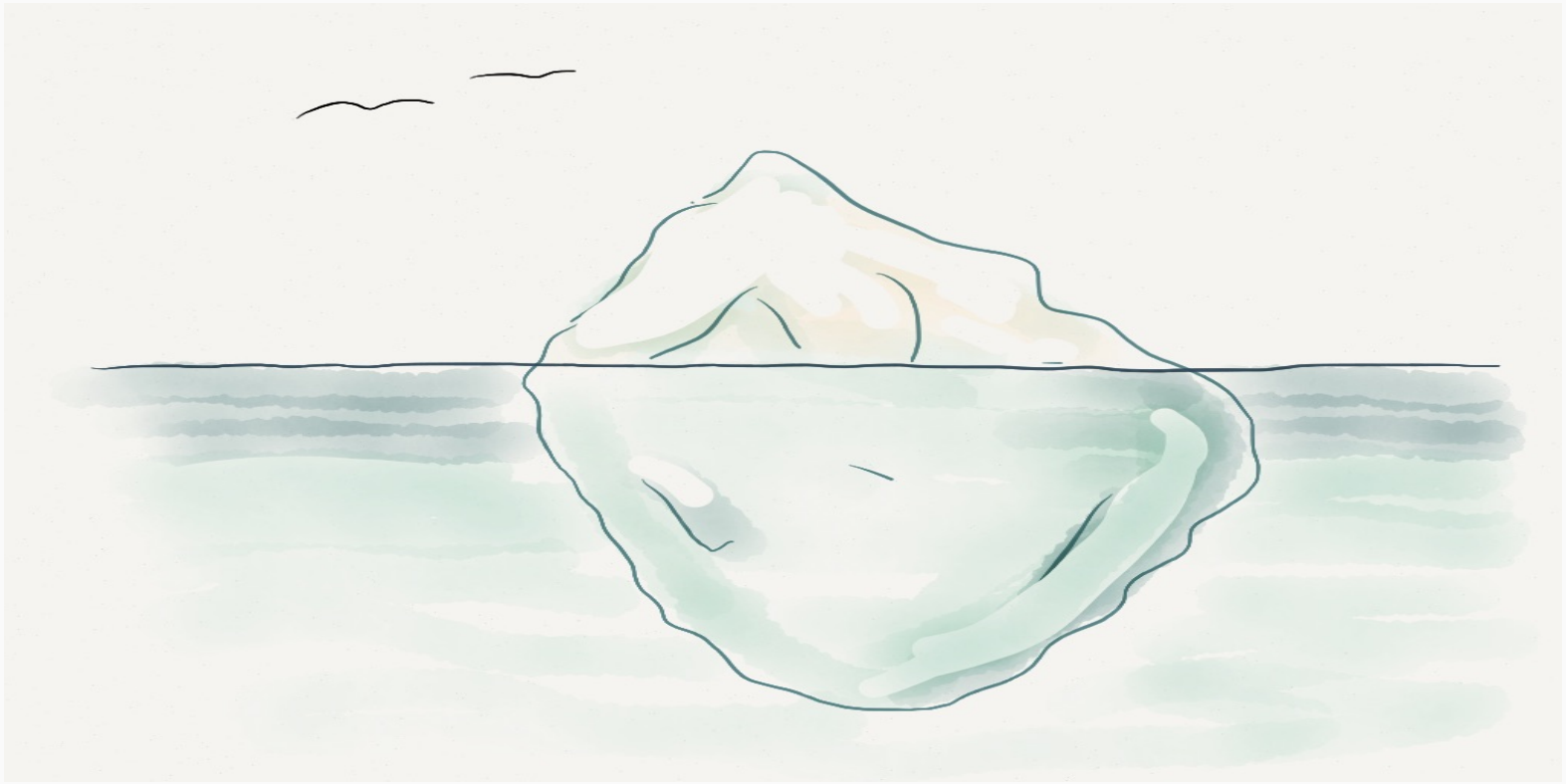


The problem

Correct proofs are tricky to write

- a lot of overhead
  (on paper and even more so in a proof assistant)
- challenging to keep track of details
- hard to understand interaction between different features
- difficulties increase with size

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]*

What are good high-level proof languages that make it easier to mechanize and maintain formal guarantees?

POPLMark – A Look Back ...

POPLMark Challenge: Mechanize System $F_<$

[Ayedemir, ... 2005]

Spotlight on

> *"type preservation and soundness, unique decomposition properties of operational semantics, proofs of equivalence between algorithmic and declarative versions of type systems."*

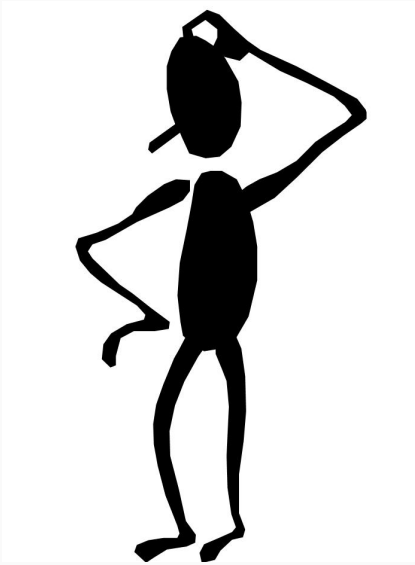
- Structural induction proofs (syntactic)
- Focus on representing and reasoning about structures with binders
- Easy to be understood; text book description (TAPL)
- Small (can be mechanized in a couple of hours or days)
- Explore more systematically different proof environments
- Explore different encoding techniques for representing bindings

POPLMark Challenge – The Good

- ✓ Popularized the use of proof assistants
- ✓ Many submitted solutions
- ✓ Good way to learn about a technique
- ✓ Mechanizing proofs is addictive!

POPLMark Challenge – The Bad

- Did we achieve *"a future where the papers in conferences such as POPL and ICFP are routinely accompanied by mechanically checkable proofs of the theorems they claim."*?
- Did we get better tool support for mechanizing proofs?



POPLMark Challenge – The Ugly

- ✗ Did not identify bugs or flaws in existing systems
- ✗ Did not inspired the development of new theoretical foundations
- ✗ Did not push existing systems to their limit

"Type soundness results are two a penny."

Andrew Pitts

Beyond the POPLMark Challenge!

Beyond the POPLMark Challenge

"The POPLMark Challenge is not meant to be exhaustive: other aspects of programming language theory raise formalization difficulties that are interestingly different from the problems we have proposed - to name a few: more complex binding constructs such as mutually recursive definitions, logical relations proofs, coinductive simulation arguments, undecidability results, and linear handling of type environments." [Aydemir et. al. 2005]

POPLMark Reloaded – Goals and Target Audience

User community including students and PL researchers:

- Teach logical relations proofs a modern way
- A good way to learn a useful and wide-spread proof technique
- Deeper understanding of the mechanics behind the technique
- Be able to grow the development to rich type theories (for example dependently typed systems)
- Understand the trade-offs in choosing a particular proof environment when tackling such a proof

Framework developers:

- Highlight features that are ideally suited for built-in support
- Highlight current shortcomings (theoretical and practical) in existing proof environments
- Signpost to advertise a given system
- Stimulate research on foundations of proof environments
- Benchmark for evaluating and comparing systems

POPLMark Reloaded:

Strong normalization for the simply-typed lambda-calculus with typed-reductions using Kripke-style logical relations

Simply Typed λ-Calculus with Type-Directed Reductions

Simply Typed λ-calculus:

$$\begin{array}{llll} \text{Terms} & M, N & ::= & x \mid \lambda x{:}A.M \mid M\ N \mid () \\ \text{Types} & A, B & ::= & \mathsf{i} \mid A \Rightarrow B \mid \mathsf{unit} \end{array}$$

Simply Typed λ-Calculus with Type-Directed Reductions

Simply Typed λ-calculus:

$$\begin{array}{llll} \text{Terms} & M, N & ::= & x \mid \lambda x{:}A.M \mid M\ N \mid () \\ \text{Types} & A, B & ::= & \mathsf{i} \mid A \Rightarrow B \mid \mathsf{unit} \end{array}$$

Type-directed reductions [Goguen'95]: $\boxed{\Gamma \vdash M \longrightarrow N : A}$

$$\frac{\Gamma \vdash \lambda x{:}A.M : A \Rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x{:}A.M)\ N \longrightarrow [N/x]M : B}\ \beta \qquad \frac{M \neq ()}{\Gamma \vdash M \longrightarrow () : \mathsf{unit}}$$

$$\frac{\Gamma \vdash M \longrightarrow M' : A \Rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M\,N \longrightarrow M'\,N : B} \qquad \frac{\Gamma \vdash M : A \Rightarrow B \quad \Gamma \vdash N \longrightarrow N' : A}{\Gamma \vdash M\,N \longrightarrow M\,N' : B}$$

$$\frac{\Gamma, x{:}A \vdash M \longrightarrow M' : B}{\Gamma \vdash \lambda x{:}A.M \longrightarrow \lambda x{:}A.M' : A \Rightarrow B}$$

Why Type-directed Reductions?

- Simplifies the study of its meta-theory.
- Concise presentation of the important issues that arise.
- Widely applicable in studying subtyping, type-preserving compilation, etc.
- Types are necessary if we want η-expansion.

$$\frac{M \neq \lambda y{:}A.M'}{\Gamma \vdash M \longrightarrow \lambda x{:}A.M\ x : A \Rightarrow B}$$

Normalization

- A term M is said to be *weakly normalising* if there is a rewrite sequence starting in M that eventually ends in a normal form
- A term M is said to be *strongly normalising* if all rewrite sequences starting in M end eventually in a normal form.

Setting the Stage: How to define strong normalization?

Often defined as an accessibility relation:

$$\frac{\forall M'.\ \Gamma \vdash M \longrightarrow N : A \Longrightarrow \Gamma \vdash N : A \in \mathsf{sn}}{\Gamma \vdash M : A \in \mathsf{sn}}$$

"the reduct analysis becomes increasingly annoying in normalization proofs for more and more complex systems." *Joachimski and Matthes [2003]*

A modular approach to strongly normalizing terms

[F. van Raamsdonk and P. Severi 1995]

- Inductive characterization of normal forms ($\Gamma \vdash M : A \in \mathsf{SN}$)
- Leads to modular proofs – on paper and in mechanizations

"the new proofs are essentially simpler than already existing ones." *F. van Raamsdonk and P. Severi*

Inductive definition of well-typed strongly normalizing terms

Neutral terms

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash x : A \in \mathsf{SNe}} \qquad \frac{\Gamma \vdash R : A \Rightarrow B \in \mathsf{SNe} \quad \Gamma \vdash M : A \in \mathsf{SN}}{\Gamma \vdash R\,M : B \in \mathsf{SNe}}$$

Normal terms

$$\frac{\Gamma \vdash R : A \in \mathsf{SNe}}{\Gamma \vdash R : A \in \mathsf{SN}} \qquad \frac{\Gamma, x{:}A \vdash M : A \Rightarrow B \in \mathsf{SN}}{\Gamma \vdash \lambda x{:}A.M : A \Rightarrow B \in \mathsf{SN}}$$

$$\frac{\Gamma \vdash M \longrightarrow_{\mathsf{SN}} M' : A \qquad \Gamma \vdash M' : A \in \mathsf{SN}}{\Gamma \vdash M : A \in \mathsf{SN}}$$

Strong head reduction

$$\frac{\Gamma \vdash N : A \in \mathsf{SN} \quad \Gamma, x{:}A \vdash M : B}{\Gamma \vdash (\lambda x.M)\ N \longrightarrow_{\mathsf{SN}} [N/x]M : B} \qquad \frac{\Gamma \vdash R \longrightarrow_{\mathsf{SN}} R' : A \Rightarrow B \quad \Gamma \vdash M : A}{\Gamma \vdash R\,M \longrightarrow_{\mathsf{SN}} R'\,M}$$

Challenge 1: Equivalence between accessibility and inductive definition of strongly normalizing terms:

$\Gamma \vdash M : A \in \mathsf{sn}$ **iff** $\Gamma \vdash M : A \in \mathsf{SN}$**.**

Strong normalization using logical relations

Strong normalization using logical relations

Definition (Reducibility Candidates: $\Gamma \vdash M \in \mathcal{R}_A$)

$$
\begin{array}{lll}
\Gamma \vdash M \in \mathcal{R}_{\mathsf{i}} & \text{iff} & \Gamma \vdash M : \mathsf{i} \in \mathsf{SN} \\
\Gamma \vdash M \in \mathcal{R}_{\mathsf{unit}} & \text{iff} & \Gamma \vdash M : \mathsf{unit} \in \mathsf{SN} \\
\Gamma \vdash M \in \mathcal{R}_{A \Rightarrow B} & \text{iff} & \Gamma \vdash M : A \Rightarrow B \text{ and} \\
 & & \quad \text{for all } N, \Delta \text{ such that } \Gamma \leq_\rho \Delta, \\
 & & \quad\quad \text{if } \Delta \vdash N \in \mathcal{R}_A \text{ then } \Delta \vdash ([\rho]M)N \in \mathcal{R}_B.
\end{array}
$$

- Contexts arise naturally
- They are necessary!
- The definition scales to dependently typed setting and stating properties about type-directed equivalence of lambda-terms.

Strong normalization using logical relations

Definition (Reducibility Candidates: $\Gamma \vdash M \in \mathcal{R}_A$)

$$
\begin{array}{lll}
\Gamma \vdash M \in \mathcal{R}_{\mathsf{i}} & \text{iff} & \Gamma \vdash M : \mathsf{i} \in \mathsf{SN} \\
\Gamma \vdash M \in \mathcal{R}_{\mathsf{unit}} & \text{iff} & \Gamma \vdash M : \mathsf{unit} \in \mathsf{SN} \\
\Gamma \vdash M \in \mathcal{R}_{A \Rightarrow B} & \text{iff} & \Gamma \vdash M : A \Rightarrow B \text{ and} \\
 & & \quad \text{for all } N, \Delta \text{ such that } \Gamma \leq_\rho \Delta, \\
 & & \quad\quad \text{if } \Delta \vdash N \in \mathcal{R}_A \text{ then } \Delta \vdash ([\rho]M)N \in \mathcal{R}_B.
\end{array}
$$

- Contexts arise naturally
- They are necessary!
- The definition scales to dependently typed setting and stating properties about type-directed equivalence of lambda-terms.

Do we really need the weakening substitution ρ?

Strong normalization using logical relations

Definition (Reducibility Candidates: $\Gamma \vdash M \in \mathcal{R}_A$)

$$
\begin{array}{lll}
\Gamma \vdash M \in \mathcal{R}_{\mathsf{i}} & \text{iff} & \Gamma \vdash M : \mathsf{i} \in \mathsf{SN} \\
\Gamma \vdash M \in \mathcal{R}_{\mathsf{unit}} & \text{iff} & \Gamma \vdash M : \mathsf{unit} \in \mathsf{SN} \\
\Gamma \vdash M \in \mathcal{R}_{A \Rightarrow B} & \text{iff} & \Gamma \vdash M : A \Rightarrow B \text{ and} \\
 & & \quad \text{for all } N, \Delta \text{ such that } \Gamma \leq_\rho \Delta, \\
 & & \quad\quad \text{if } \Delta \vdash N \in \mathcal{R}_A \text{ then } \Delta \vdash ([\rho]M)N \in \mathcal{R}_B.
\end{array}
$$

- Contexts arise naturally
- They are necessary!
- The definition scales to dependently typed setting and stating properties about type-directed equivalence of lambda-terms.

Do we really need to model terms in a "local" context and use Kripke-style context extensions?

Challenge 2: Strong normalization for simply typed λ-calculus

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

Challenges in the proof(s)

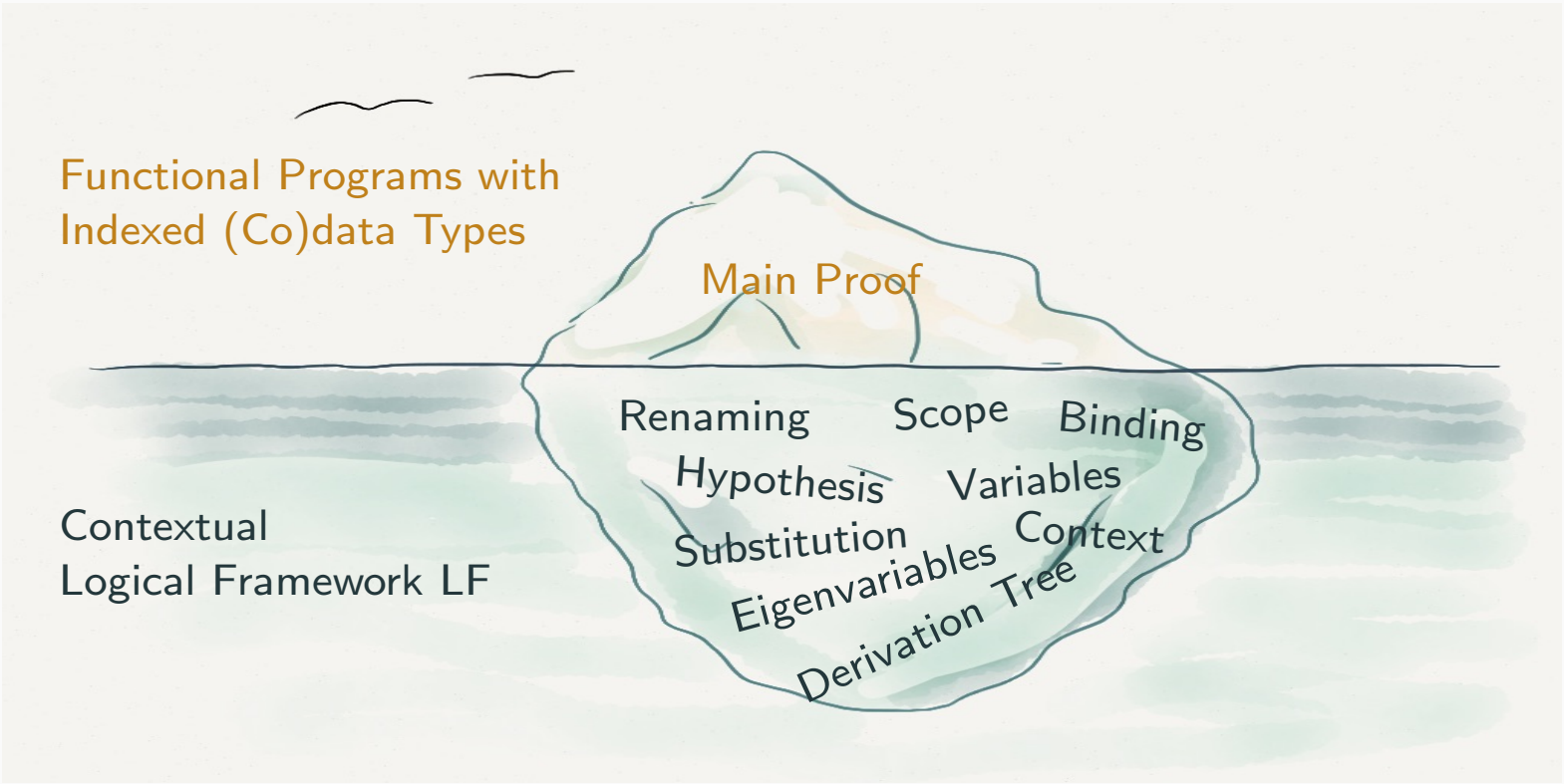
- Definitions use well-typed terms
- Stratified definitions for reducibility candidates
  (not strictly positive!)
- Simultaneous substitutions and renamings
- Basic infrastructure
  - Substitution properties about terms
  - Weakening and Strengthening of type-directed reductions
  - Weakening, Exchange, and Strengthening for typing
  - Weakening and Exchange strongly normalizing terms
  - Renaming, Anti-renaming, Substitution properties for strongly normalizing terms
- Induction principles
  (not necessarily on direct sub-derivation)

Towards solving the challenge problems

On the way

- Coq mechanization using Autosubst
  – add pictures of people involved
- Abella
  – add pictures of people involved
- Agda using well-scoped de Bruijn indices
  – add pictures of people involved
- Beluga

Beluga: Programming and Proof Environment



- Below the surface: Support for key concepts based on Contextual LF [TOCL'08,POPL'08,LFMTP'13, ESOP'17, ...]
- Above the surface: (Co)Inductive Proofs
  as (Co)Recursive Programs using (Co)pattern Matching
  [POPL'08,IJCAR'10, POPL'12,POPL'13,CADE'15,ICFP'16, ...]

A Quick Guided Tour

Demo

A confession ...

What we've done in Beluga so far

- SN proof with inductive definition
- Set up scales to sums and products
- Built-in support for simultaneous substitutions leads to compact proofs

Still to do ...

A confession

Thanks, Aliya.

Lessons learned – The Good

Elegant.

- HOAS is great!
- Built-in support for simultaneous substitutions and renamings is great!
- Take advantage of dependent types to model intrinsically typed terms, typed-reductions, typed SN, etc.
- Modular, scales
- Compact

Lessons learned – The Bad and Ugly

- How to pattern matching on renamings $[\rho]M$?
- Interactive Mode to develop proofs needs work
- Termination checker not powerful enough
-

Confession: we are missing implementation or anti-renaming lemmas ...

Isn't proving strong normalization in a proof assistant an old hat?

Just following Girard's "Proofs and Types"

Characteristic Features:

- Terms are not well-scoped or well-typed
- Candidate relation is untyped and does not enforce well-scoped terms

$\Longrightarrow$ does not scale to typed-directed evaluation or equivalence

$\Longrightarrow$ today we have better techniques to structure proof

Beyond strong normalization ...

Additional Challenge Problems

- Weak normalization
  (good starting point)
- Type-directed algorithmic equality
  (done in Beluga)
- Adding η-expansion
- Normalization of System F
  (excellent suggestion – currently beyond what Beluga can do)

Evaluation Criteria

A Call for Action

- Be part of formulating and tackling the challenge
- Choose your favorite proof assistant and complete the challenge

Let's get started...

Let's get started...talk to me for the challenge problem set up

Thank you!