POPLMark Reloaded!

Andreas Abel [1] Alberto Momigliano [2] Brigitte Pientka [3]

[1]Department of Computer Science and Engineering, Gothenburg University, Sweden

[2]DI, Università degli Studi di Milano, Italy

[3]School of Computer Science, McGill University, Montreal, Canada

September 11, 2017

POPLMark Reloaded: A new benchmark for mechanizing meta-theory of programming languages

Strong normalization of the simply-typed lambda-calculus using Kripke-style logical relations.

Why do we need a (new) benchmark?

Before 2005: A Brief Incomplete History

- Isabelle [1986], Coq[1989], Alf/Agda 1 [1990 – 2007], Lego [1995/98], Elf/Twelf[1993/1998], . . .
- Case studies: Type Soundness, Church Rosser, Cut-elimination, Compilation, . . .
- Focus on reasoning about formal systems by structural induction; modelling variable bindings; assumptions; etc.
- Canonical example: Type soundness
- Some normalization proofs:
  - Altenkirch, SN for System F in Lego [TLCA 1993]
  - Barras/Werner, SN for CoC in Coq [1997]
  - C. Coquand, NbE for $\lambda\sigma$ in ALFA [1999]
  - Berghofer, WN for STL in Isabelle [TYPES 2004]
  - Abel, WN/SN for STL in Twelf [LFM 2004]

POPLMark Challenge: Mechanize System $F_<$ [2005]

- Spotlight on

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

- 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

POPLMark Challenge: Looking back

- ✓ Popularized the use of proof assistants
- ✓ Many submitted solutions
- ✓ Explored different techniques for representing bindings
- ✓ Good way to learn about a technique / proof assistant

POPLMark Challenge: Looking back

- ✓ Popularized the use of proof assistants
- ✓ Many submitted solutions
- ✓ Explored different techniques for representing bindings
- ✓ Good way to learn about a technique / proof assistant

- ? Long Term Goal: "a future where the papers in conferences such as POPL and ICFP are routinely accompanied by mechanically checkable proofs of the theorems they claim."
- ? Better understanding of the theoretical foundations of proof environments

POPLMark Challenge: Looking back

- ✓ Popularized the use of proof assistants
- ✓ Many submitted solutions
- ✓ Explored different techniques for representing bindings
- ✓ Good way to learn about a technique / proof assistant

- ? Long Term Goal: "a future where the papers in conferences such as POPL and ICFP are routinely accompanied by mechanically checkable proofs of the theorems they claim."
- ? Better understanding of the theoretical foundations of proof environments

- ✗ Inspired the development of new theoretical foundations
- ✗ Better tool support

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: Goal

Benchmark problems that

- Push the state of the art in the area and outline new areas of research
- Compare systems and mechanized proofs qualitatively
- Understand what infrastructural parts should be generically supported and factored
- Find bugs in existing proof assistants
- Highlight theoretical limitations of existing proof environments
- Highlight practical limitations of existing proof environments

Why pick strong normalization for simply-typed lambda-calculus using Kripke-style logical relations?

Why pick strong normalization for simply-typed lambda-calculus using Kripke-style logical relations?

In particular:

> We can prove SN without (Kripke-style) logical relations and we've already done it.

Witness 1: Lego [Altenkirch'93]

... "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
  $\Longrightarrow$ maybe better techniques to modularize and structure proof

Witness 2: Abella, ATS/HOAS

. . . "following Girard's Proofs and Types"

Witness 2: Abella, ATS/HOAS

... "following Girard's Proofs and Types"

- Strictly speaking:

  SN for simply-typed λ-calculus plus one constant.

- Adding a constant significantly simplifies the proof
- Reducibility of terms only defined on closed terms
- Strictly speaking:

  Show that SN for simply-typed λ-calculus plus one constant implies also SN for open simply-typed λ-terms

More Witnesses . . .

- Berghofer : Program extraction from a proof of weak normalization using Isabelle [2004]
  $\Longrightarrow$ Uses de Bruijn encoding (not well-scoped or well-typed)
  $\Longrightarrow$ "Compact" mechanization (800 lines)

More Witnesses ...

- Berghofer : Program extraction from a proof of weak normalization using Isabelle [2004]
  $\Longrightarrow$ Uses de Bruijn encoding (not well-scoped or well-typed)
  $\Longrightarrow$ "Compact" mechanization (800 lines)

- Berger et al. [TLCA'93]: Extraction of a normalization by evaluation using strong evaluation in Minlog
  $\Longrightarrow$ Uses well-scoped de Bruijn encoding
  $\Longrightarrow$ Domain theoretic semantics

More Witnesses . . .

- Berghofer : Program extraction from a proof of weak normalization using Isabelle [2004]
  $\Longrightarrow$ Uses de Bruijn encoding (not well-scoped or well-typed)
  $\Longrightarrow$ "Compact" mechanization (800 lines)

- Berger et al. [TLCA'93]: Extraction of a normalization by evaluation using strong evaluation in Minlog
  $\Longrightarrow$ Uses well-scoped de Bruijn encoding
  $\Longrightarrow$ Domain theoretic semantics

- Doczkal, Schwinghammer [LFMTP'09]: Mechanization of Strong Normalization Proof for Moggis Computational Metalanguage in Isabelle/Nominal
  $\Longrightarrow$ Use of nominals avoids Kripke-style formulation

Why Kripke-style?

- Kripke-style extensions cannot be avoided when we attempt to prove properties about type-directed evaluation
  (see for example mechanizations of Crary's proof of completenes of algorithmic equality for LF)

- We want to keep the benchmark problem simple, but it should exhibit features that allow us to scale systems to more complex problems.

Setting the Stage: Simply Typed Lambda-Calculus

Terms	M, N	$::=$	$x \mid \lambda x{:}T.M \mid M\ N$
Types	T, S	$::=$	$B \mid T \Rightarrow S$
Context	Γ	$::=$	$\cdot \mid \Gamma, x{:}T$
Subs	σ	$::=$	$\epsilon \mid \sigma, N/x$

$\boxed{\Gamma \vdash M : T}$ Term M has type T in context Γ

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma, x : T \vdash M : S}{\Gamma \vdash (\lambda x{:}T.M) : (T \Rightarrow S)} \qquad \frac{\Gamma \vdash M : (T \Rightarrow S) \quad \Gamma \vdash N : T}{\Gamma \vdash (M\ N) : S}$$

Setting the Stage: Simply Typed Lambda-Calculus

$$
\begin{array}{llcl}
\text{Terms} & M, N & ::= & x \mid \lambda x{:}T.M \mid M\ N \\
\text{Types} & T, S & ::= & B \mid T \Rightarrow S \\
\text{Context} & \Gamma & ::= & \cdot \mid \Gamma, x{:}T \\
\text{Subs} & \sigma & ::= & \epsilon \mid \sigma, N/x
\end{array}
$$

$\boxed{\Gamma \vdash M : T}$ Term M has type T in context Γ

$$
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma, x : T \vdash M : S}{\Gamma \vdash (\lambda x{:}T.M) : (T \Rightarrow S)} \qquad \frac{\Gamma \vdash M : (T \Rightarrow S) \quad \Gamma \vdash N : T}{\Gamma \vdash (M\ N) : S}
$$

Implement well-typed lambda-terms any way you like!
Intrinsically typed, explicit typing, explicit typing context, HOAS-style, Nominal, de Bruijn, ...

Setting the Stage: Evaluation

$\boxed{\Gamma \vdash M \longrightarrow M'}$ Term M steps to term M' in context Γ

$$\frac{\Gamma, x{:}T \vdash M \longrightarrow M'}{\Gamma \vdash \lambda x{:}T.M \longrightarrow \lambda x{:}T.M'} \qquad \frac{}{\Gamma \vdash (\lambda x{:}T.M)\ N \longrightarrow [N/x]M}$$

$$\frac{\Gamma \vdash M \longrightarrow M'}{\Gamma \vdash M\ N \longrightarrow M'\ N} \qquad \frac{\Gamma \vdash N \longrightarrow N'}{\Gamma \vdash M\ N \longrightarrow M\ N'}$$

Remark: We chose to make Γ explicit in the evaluation rules; **this is not a requirement!** – But your implementation of the rules must allow for evaluating terms with free variables.

Setting the Stage: Reducibility

Reducibility must be defined on well-typed open terms!

Setting the Stage: Reducibility

Reducibility must be defined on well-typed open terms!

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

$$
\begin{array}{lll}
\Gamma \vdash M \in B & \text{iff} & \Gamma \vdash M : B \text{ and } \Gamma \vdash M \in \mathsf{sn} \\
\Gamma \vdash M \in T \Rightarrow S & \text{iff} & \Gamma \vdash M : T \Rightarrow S \text{ and} \\
 & & \text{for all } N, \Delta \text{ such that } \Gamma \leq_\rho \Delta, \\
 & & \quad \text{if } \Delta \vdash N \in \mathcal{R}_T \text{ then } \Delta \vdash ([\rho]M)\ N \in \mathcal{R}_S.
\end{array}
$$

- Contexts arise naturally when we want to state properties about well-typed terms and we want to be precise.
- The definition scales to dependently typed setting and stating properties about type-directed equivalence of lambda-terms.

Setting the Stage: Reducibility

Reducibility must be defined on well-typed open terms!

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

$$
\begin{array}{lll}
\Gamma \vdash M \in B & \text{iff} & \Gamma \vdash M : B \text{ and } \Gamma \vdash M \in \mathsf{sn} \\
\Gamma \vdash M \in T \Rightarrow S & \text{iff} & \Gamma \vdash M : T \Rightarrow S \text{ and} \\
& & \text{for all } N, \Delta \text{ such that } \Gamma \leq_\rho \Delta, \\
& & \quad \text{if } \Delta \vdash N \in \mathcal{R}_T \text{ then } \Delta \vdash ([\rho]M)\ N \in \mathcal{R}_S.
\end{array}
$$

- Contexts arise naturally when we want to state properties about well-typed terms and we want to be precise.
- 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?

Setting the Stage: Reducibility

Reducibility must be defined on well-typed open terms!

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

$$
\begin{array}{lll}
\Gamma \vdash M \in B & \text{iff} & \Gamma \vdash M : B \text{ and } \Gamma \vdash M \in \mathsf{sn} \\
\Gamma \vdash M \in T \Rightarrow S & \text{iff} & \Gamma \vdash M : T \Rightarrow S \text{ and} \\
 & & \text{for all } N, \Delta \text{ such that } \Gamma \leq_\rho \Delta, \\
 & & \quad \text{if } \Delta \vdash N \in \mathcal{R}_T \text{ then } \Delta \vdash ([\rho]M)\ N \in \mathcal{R}_S.
\end{array}
$$

- Contexts arise naturally when we want to state properties about well-typed terms and we want to be precise.
- 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?

Setting the Stage: Strong Normalization

Often defined as:

$$\frac{\forall M'.\, \Gamma \vdash M \ \longrightarrow\ M' \Longrightarrow \Gamma \vdash M' \in \mathsf{SN}}{\Gamma \vdash M \in \mathsf{SN}}$$

Setting the Stage: Strong Normalization

Often defined as:

$$\frac{\forall M'.\ \Gamma \vdash M \ \longrightarrow\ M' \Longrightarrow \Gamma \vdash M' \in \mathsf{SN}}{\Gamma \vdash M \in \mathsf{SN}}$$

Alternative approach (R. Matthes and F. Joachimski, AML 2003)

- Inductive characterization of normal forms ($\Gamma \vdash M \in \mathsf{sn}$)
- Normalization proof is by induction on normal forms and type expressions
- Leads to modular proofs – on paper and in mechanizations
- Show: $\Gamma \vdash M \in \mathsf{SN}$ iff $\Gamma \vdash M \in \mathsf{sn}$.

Why do we think this is an interesting case study?

- Richer induction principles needed than just structural induction based on sub-derivations
- Stratified definitions for reducibility candidates
- Comparison and trade-offs when modelling well-scoped and well-typed terms
- Good way to teach logical relations proofs
  $\Longrightarrow$ maybe extend it to products and sums

A Call for Action

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

Last but not least: Propose a different challenge!