Andreas Abel\textsuperscript{1} \quad Alberto Momigliano\textsuperscript{2} \quad Brigitte Pientka\textsuperscript{3}

\textsuperscript{1}Department of Computer Science and Engineering, Gothenburg University, Sweden

\textsuperscript{2}DI, Università degli Studi di Milano, Italy

\textsuperscript{3}School of Computer Science, McGill University, Montreal, Canada

September 7, 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

- 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]
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
  \[\implies\] does not scale to typed-directed evaluation or equivalence
  \[\implies\] maybe better techniques to modularize and structure proof
Witness 2: Abella, ATS/HOAS

... “following Girard’s Proofs and Types”
... “following Girard’s Proofs and Types”

- Strictly speaking:
  
  $\text{SN for simply-typed } \lambda\text{-calculus plus one constant.}$

- Adding a constant significantly simplifies the proof

- Reducibility of terms only defined on closed terms

- Strictly speaking:

  $\text{Show that SN for simply-typed } \lambda\text{-calculus plus one constant implies also SN for open simply-typed } \lambda\text{-terms}$
Berghofer: Program extraction from a proof of weak normalization using Isabelle [2004]

⇒ Uses de Bruijn encoding (not well-scoped or well-typed)

⇒ “Compact” mechanization (800 lines)
More Witnesses . . .

- Berghofer: Program extraction from a proof of weak normalization using Isabelle [2004]
  \[ \Rightarrow \text{Uses de Bruijn encoding (not well-scoped or well-typed)} \]
  \[ \Rightarrow \text{“Compact” mechanization (800 lines)} \]

- Berger et al. [TLCA’93]: Extraction of a normalization by evaluation using strong evaluation in Minlog
  \[ \Rightarrow \text{Uses well-scoped de Bruijn encoding} \]
  \[ \Rightarrow \text{Domain theoretic semantics} \]
More Witnesses . . .

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

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

- Doczkal, Schwinghammer [LFMTP’09]: Mechanization of Strong Normalization Proof for Moggis Computational Metalanguage in Isabelle/Nominal
  \[\implies\] 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 completeness 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 \[ M, N ::= x | \lambda x: T. M | M N \]

Types \[ T, S ::= B | T \Rightarrow S \]

Context \[ \Gamma ::= \cdot | \Gamma, x: T \]

Subs \[ \sigma ::= \epsilon | \sigma, N/x \]

\[ \Gamma \vdash M : T \] Term \( M \) has type \( T \) in context \( \Gamma \)

\[
\begin{align*}
\Gamma \vdash x : T & \quad \Gamma, x : T \vdash M : S \\
\Gamma, x : T \vdash M : S & \quad \Gamma \vdash (\lambda x: T. M) : (T \Rightarrow S) \\
\Gamma \vdash M : (T \Rightarrow S) & \quad \Gamma \vdash N : T \\
\Gamma \vdash (M N) : S
\end{align*}
\]
Setting the Stage: Simply Typed Lambda-Calculus

Terms \( M, N \ ::= \ x | \lambda x: T. M | M \ N \)

Types \( T, S \ ::= \ B | T \Rightarrow S \)

Context \( \Gamma \ ::= \cdot | \Gamma, x: T \)

Subs \( \sigma \ ::= \epsilon | \sigma, N/x \)

\[ \Gamma \vdash M : T \] Term \( M \) has type \( T \) in context \( \Gamma \)

\[
\begin{align*}
\Gamma \vdash x : T \in \Gamma & \quad \Gamma, x : T \vdash M : S & \quad \Gamma \vdash (\lambda x: T. M) : (T \Rightarrow S) \\
\Gamma \vdash N : T & \quad \Gamma \vdash (M \ N) : S
\end{align*}
\]

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

Term $M$ steps to term $M'$ in context $\Gamma$

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

$$
\frac{}{\Gamma \vdash M \rightarrow M'}
\frac{}{\Gamma \vdash M N \rightarrow M' N}
\frac{}{\Gamma \vdash M N \rightarrow M' N'}
$$

Remark: We chose to make $\Gamma$ explicit in the evaluation rules; this is not a requirement! – But your implementation of the rules must allow for evaluating terms with free variables.
Reducibility must be defined on well-typed open terms!
Reducibility must be defined on well-typed open terms!

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

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

- 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: \( \Gamma \vdash M \in \mathcal{R}_B \))**

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

- 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 \( \rho \)?
Reducibility must be defined on well-typed open terms!

Definition (Reducibility Candidates: $\Gamma \vdash M \in R_B$)

$\Gamma \vdash M \in B$ iff $\Gamma \vdash M : B$ and $\Gamma \vdash M \in \text{sn}$

$\Gamma \vdash M \in T \Rightarrow S$ iff $\Gamma \vdash M : T \Rightarrow S$ and

for all $N, \Delta$ such that $\Gamma \leq_{\rho} \Delta$,

if $\Delta \vdash N \in R_T$ then $\Delta \vdash ([\rho]M) N \in R_S$.

- 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?
Often defined as:

\[
\forall M'. \quad \Gamma \vdash M \rightarrow M' \quad \Rightarrow \quad \Gamma \vdash M' \in SN
\]

\[
\Gamma \vdash M \in SN
\]
Setting the Stage: Strong Normalization

Often defined as:

\[
\forall M'. \; \Gamma \vdash M \rightarrow M' \implies \Gamma \vdash M' \in SN \\
\Gamma \vdash M \in SN
\]

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

- Inductive characterization of normal forms
- Normalization proof is by induction on normal forms and type expressions
- Leads to modular proofs – on paper and in mechanizations
Why do we thing 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
  \[\Rightarrow\] 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!