Benchmarks for mechanized meta-theory
A very personal and partial view

Alberto Momigliano

Dagstuhl, Oct 17, 2016
Loosely based on joint work with
Amy Felty and Brigitte Pientka
Disclaimer

- These are half-baked thoughts cooked up on the plane, offered here only to get a discussion going.
- As Donald Rumsfeld used to say:
  - There are things I know I don’t know.
  - There are things I don’t know I don’t know.
- So feel free to interject at any time.
Introduction

- Benchmarks in theorem proving are useful, make the state of the art progress or at least take stock
  - automating FOL: TPTP [Sutcliffe, JAR 2009]
  - higher-order extension: THF0 [Benzmüller et al, IJCAR 2008]
  - SATLIB [Hoos & Stützle, SAT 2000]

- The situation is less satisfactory for proof assistants, where each system comes with its own set of examples/libraries, some of them, as we well know, gigantic.

- Not surprising, since we’re potentially addressing the whole realm of maths

- So, let’s pull the breaks and limit ourselves here to the meta-theory of deductive systems, such as the ones in Programming Languages theory
Benchmarks for PLT

- Where do we stand?
  - There are obvious differences with say TPLP, qualitative vs quantitative comparisons

- In a dark and stormy night of 2005 the POPLMark Challenge appeared. And then?
  - Lots of clever solutions (mostly incomplete) and to be fair almost all using existing technology
  - What kind of impact on the communities (PL and LF researchers)?
Impact on PL research

- More papers at POPL/ICFP come with at least a partial formalization
  - Sewell POPL 14: “Around 10% of submissions were completely formalised, slightly more partially formalised”
- Formalizations are evaluated (http://www.artifact-eval.org/)
- Can I have an hallelujah? Do we feel, as logical framework people vindicated? Does it make a difference for acceptance (or for the betterment of science)?
  - Sewell again “acceptance rates for these were in line with those for submissions as a whole.”
Impact on PL research

- Significant success of courses such as “Software Foundations” and to a lesser extent “Concrete Semantics” has been forming new generations of PL people who consider proof assistants as part of the tools belt (OK, it’s mostly Coq but still)
- Huge success of projects where formalization is paramount such as CompCert, Relaxed Concurrency Models, SEL4 etc.
- Significant fundings for new ones (REMS, DeepSpec, SECOMP, RustBelt)
Impact on logical frameworks

- Did they evolve in the last 10 years as a response or on their own accord? Feel free to disagree
  - **HOAS**: Twelf stood still, Abella and Beluga were born out of research (nabla and fixed point logics, CMTT) stemming from the early 2000.
    - to be generous, one could say that Abella 2.0’s generalization of SL logic to higher-order owes to Pientka’s solution of POPLMark 1.a
    - They still are systems with very limited user-base outside developers
  - **Nominal**: the Isabelle package turned into Nominal2, but more to have a better foundation for equivariance and to handle multiple binders
    - From the traffic on the mailing list, I’d say Nominal Isabelle has gone quiet, although it has some power users not deterred by the amount of low-level proofs one has still to handle
    - Nominal techniques in Coq: I don’t follow HTT (too hard for me), so don’t know
Coq’s masterplan for world domination

- Is a monopoly in PL theory necessarily a bad thing? Dale Miller is proposing a *marketplace* for proofs, is Coq the new Facebook killing it in its infancy?
  - Large user base
  - Very active development
- What about libraries for binders: are they used? Are people happy with that?
  - *Locally nameless* from the “Engineering Formal Metatheory” paper
  - Smolka’s *Autosubst*
  - *Hybrid* (no, and I understandably so, until we make it usable by others).
Is my obsession with binders justified?

- Well, not just mine (three words: Harper, Honsell & Milner)
- Binders are ubiquitous in PLT as well as obnoxious and need to be handled simply, correctly and generically.
- Still, a fair amount of formalization stays (apparently successfully so) away from binders.
- “LEM does not care about binders, and we’re happy to keep it that way” (Peter Sewell, my recollection 2014)
- Not to name names (got it?), Leroy’s celebrated “Coinductive big-step operational semantics” uses named not $\alpha$-equivalent syntax, non-capture avoiding substitutions, and so does Software Foundations.
- They know what they’re doing. Maybe they’re right?
In the 90’s we were happy about proving type soundness for PCF and friends – graduated to memory safety following PCC

Then we got to the Church-Rosser theorem – at least it deals with open not closed terms

POPLMark, we already saw.

We (Amy Felty, Brigitte Pientka and myself) proposed some (by design) simple benchmarks to stress systems for reasoning in a context – see the appendix of this talk

For example, the equivalence of declarative and algorithmic ($\alpha$)-equality between lambda terms
Benchmarks, the present (and future?)

- Many reliability and security properties of software depend upon some notion of **program equivalence**:
  - information-flow security (confinement, non-interference). More in general:
  - Bisimulation between (concurrent) systems
  - full abstraction of program transformation within the same language or from high to low level code. More in general:
    - compiler correctness (too many advances to even mention), most of them formalized

- A unifying theme is the emerging, pervasive use of **logical relations**, in particular step-indexed.

- Seems to be prevailing over **coinductive** techniques, perhaps because the latter are less supported in proof assistants (wrt inductive ones, I mean)
Step-indexed logical relations

- People have formalized them quite often by now, so what’s the fuss?
- Some do, yes, but it’s still challenging to do it right:
  - The logical relation is not immediate to be accepted as an inductive definition
  - It involves a bit of arithmetic reasoning, and this may be annoying for hyper-pure frameworks

“definitions and proofs have a tendency to become cluttered with extra indices and even arithmetic, which are really playing the role of construction lines” (Benton and Hur, “Step-Indexing: The Good, the Bad and the Ugly”)

An example: “A verified framework for higher-order uncurrying optimization (Dargaye & Leroy, 2010)

- Interesting as it uses coinduction to encode cyclic closure (the Milner-Tofte trick)
- But they use closure to avoid substitutions . . .
- it leaves open what to do with divergent source programs

Still a lot of work for a source language that is the untyped \( \lambda \)-calculus with \texttt{letrec}

A paper such as “Typed Closure Conversion Preserves Observational Equivalence” (Ahmed et al. 2008), where the source language is system F with existential and recursive types is way harder – and I haven’t look at the more recent stuff about multi-language semantics
Other directions (please contribute)

- Interesting mix of (co)induction and (co)recursion (e.g. papers by Nakata-Uustalu among others)
- Work on relational reasoning where polymorphism really is a plus (Howe’s method in its generality and more in general Soren Lassen’s thesis)
- Benchmarks where object logics make heavy use of constraint domains (see the Twelf/clp-examples directory, work by Ivan Scagnetto and al. on LF + oracles)
One slide on ORBI

- Benchmarks to be communicated need a language

  ▶ ORBI is designed to be:
    ★ human-readable
    ★ easily machine-parsable
    ★ uniform
    ★ flexible and extensible

  ▶ Currently oriented toward supporting the following systems:
    ★ Twelf
    ★ Beluga
    ★ Abella
    ★ Hybrid

without hopefully precluding other current and future systems supporting HOAS, and eventually supporting other representation techniques such as nominal.
Related Work

- The above libraries for ATP
- Very little about **inductive** problems
- Ott and LEM as front-ends
- LF as a common ground, e.g.,
  - Logosphere (http://www.logosphere.org)
  - SASyLF [Aldrich et al, 2008]
  - Modularity in LF specifications [Rabe & Schürmann]
- Why3 (http://why3.lri.fr), a software verification platform providing a front-end to third-party theorem provers.
- Environments for programming language descriptions
  - PLT-Redex [Felleisen et al, 2009]
  - The K framework [Roșu & Șerbănuță, JLAP, 2010]
- Handling and sharing of mathematical content
Appendix

Some benchmarks from:

Untyped Lambda Terms

Types  \( A, B \ ::= \alpha | arr A B \)

Terms  \( M ::= x \ lam x. M | app M_1 M_2 \)

Well-Formed Terms: \((is\_tm\ M)\)

\[
\begin{align*}
\frac{is\_tm \ x \in \Gamma}{\Gamma \vdash is\_tm \ x} & \quad tm_v \\
\frac{\Gamma, is\_tm \ x \vdash is\_tm \ M}{\Gamma \vdash is\_tm \ (lam \ x. M)} & \quad tm_l \\
\frac{\Gamma \vdash is\_tm \ M_1 \quad \Gamma \vdash is\_tm \ M_2}{\Gamma \vdash is\_tm \ (app \ M_1 \ M_2)} & \quad tm_a
\end{align*}
\]

Equality of Lambda Terms

Algorithmic Equality: \((\text{aeq } M \ N)\)

- **aeq \(x \ x \in \Gamma\)**
  \[\frac{}{\Gamma \vdash \text{aeq } x \ x} \quad \text{ae}_v\]

- **\(\Gamma, \text{is\_tm } x; \text{aeq } x \ x \vdash \text{aeq } M \ N\)**
  \[\frac{}{\Gamma \vdash \text{aeq } (\text{lam } x. \ M) \ (\text{lam } x. \ N)} \quad \text{ae}_l\]

- **\(\Gamma \vdash \text{aeq } M_1 \ N_1\)
  \(\Gamma \vdash \text{aeq } M_2 \ N_2\)**
  \[\frac{}{\Gamma \vdash \text{aeq } (\text{app } M_1 \ M_2) \ (\text{app } N_1 \ N_2)} \quad \text{ae}_a\]

Contexts

- **\(S_x := \text{is\_tm } x\)**
- **\(S_{xa} := \text{is\_tm } x; \text{aeq } x \ x\)**

Note: \(S_{xa}\) is a basic linear extension of \(S_x\).
Benchmark: Basic Linear Context Extension

Theorem (Admissibility of Reflexivity)
If $\Phi_{xa} \vdash is\_tm M$ then $\Phi_{xa} \vdash aeq M M$.

Theorem (Admissibility of Symmetry and Transitivity)
1. If $\Phi_{xa} \vdash aeq M N$ then $\Phi_{xa} \vdash aeq N M$.
2. If $\Phi_{xa} \vdash aeq M L$ and $\Phi_{xa} \vdash aeq L N$ then $\Phi_{xa} \vdash aeq M N$. 
The Polymorphic Lambda Calculus

Types \( A, B \ ::= \alpha \mid \text{arr} A B \mid \text{all} \alpha. A \)

Terms \( M \ ::= x \mid \text{lam} x. M \mid \text{app} M_1 M_2 \mid \text{tlam} \alpha. M \mid \text{tapp} M A \)

Rules

Well-formedness of Types \( (\text{is\_tp} A) \)

Well-formedness of Terms \( (\text{is\_tm} M) \)

Equality of Types \( (\text{atp} A B) \)

Equality of Terms \( (\text{aeq} M N) \)

Context Schemas

\( S_\alpha ::= \text{is\_tp} \alpha \)

\( S_{\alpha x} ::= \text{is\_tp} \alpha + \text{is\_tm} x \)

\( S_{\text{atp}} ::= \text{is\_tp} \alpha; \text{atp} \alpha \alpha \)

\( S_{\text{aeq}} ::= \text{is\_tp} \alpha; \text{atp} \alpha \alpha + \text{is\_tm} x; \text{aeq} x x \)
Linear Context Extensions with Alternatives

Theorem (Admissibility of Reflexivity for Types)
If $\Phi_{atp} \vdash \text{is}_{-\text{tp}} A$ then $\Phi_{atp} \vdash \text{atp} A A$.

Theorem (Admissibility of Reflexivity for Terms)
If $\Phi_{aeq} \vdash \text{is}_{-\text{tm}} M$ then $\Phi_{aeq} \vdash \text{aeq} M M$. 
Non-linear Context Extensions

Declarative Equality of the Untyped Lambda Calculus

\[\ldots\]

\[\Gamma \vdash \text{deq } M \equiv M \quad \text{de}_r\]

\[\Gamma \vdash \text{deq } M \equiv L \quad \Gamma \vdash \text{deq } L \equiv N \quad \Gamma \vdash \text{deq } L \equiv N \quad \text{de}_t\]

\[\Gamma \vdash \text{deq } N \equiv M \quad \Gamma \vdash \text{deq } M \equiv N \quad \text{de}_s\]

Context Schemas

\[S_{xd} ::= \text{is\_tm } x; \text{deq } x \equiv x\]

\[S_{da} ::= \text{is\_tm } x; \text{deq } x \equiv x; \text{aeq } x \equiv x\]

Theorem (Completeness)

If \(\Phi_{da} \vdash \text{deq } M \equiv N\) then \(\Phi_{da} \vdash \text{aeq } M \equiv N\).
Order

Theorem (Pairwise Substitution)
If $\Phi_{xa}, \text{is\_tm\,} x; \ aeq\ x\ x \vdash\ aeq\ M_1\ M_2$ and $\Phi_{xa} \vdash\ aeq\ N_1\ N_2$, then $\Phi_{xa} \vdash\ aeq\ ([N_1/x]M_1)\ ([N_2/x]M_2)$.

Uniqueness

Terms

$M ::= x \mid \text{lam\,} x.\ M \mid \text{app}\ M_1\ M_2$

Types

$A ::= i \mid \text{arr}\ A\ B$

Context Schema

$S_t ::= \text{is\_tm}\ x;\ x:A$

Theorem (Type Uniqueness)
If $\Phi_t \vdash\ M : A$ and $\Phi_t \vdash\ M : B$ then $A = B$. 
Substitution

Parallel Reduction

\[
\begin{align*}
\frac{x \rightsquigarrow x \in \Gamma}{\Gamma \vdash x \rightsquigarrow x} & \quad pr_v \\
\frac{\Gamma, \text{is}_\text{tm} x; x \rightsquigarrow x \vdash M \rightsquigarrow N}{\Gamma \vdash \lambda x. M \rightsquigarrow \lambda x. N} & \quad pr_l \\
\frac{\Gamma, \text{is}_\text{tm} x; x \rightsquigarrow x \vdash M \rightsquigarrow M'}{\Gamma \vdash N \rightsquigarrow N'} & \quad pr_\beta \\
\frac{\Gamma \vdash (\text{app} (\lambda x. M) N) \rightsquigarrow [N'/x]M'}{\Gamma \vdash (\text{app} M N) \rightsquigarrow (\text{app} M' N')} & \quad pr_a
\end{align*}
\]

Context Schemas

\[
\begin{align*}
S_r & := \text{is}_\text{tm} x; x \rightsquigarrow x \\
S_{rt} & := \text{is}_\text{tm} x; x \rightsquigarrow x; x:A
\end{align*}
\]
Substitution (Continued)

Lemma (Substitution)
If $\Phi_t, \text{is}_\text{tm} x; x:A \vdash M : B$ and $\Phi_t \vdash N : A$, then $\Phi_t \vdash [N/x]M : B$.

Theorem (Type Preservation for Parallel Reduction)
If $\Phi_{rt} \vdash M \rightsquigarrow N$ and $\Phi_{rt} \vdash M : A$, then $\Phi_{rt} \vdash N : A$.

Context Schema
$S_{\alpha t} ::= \text{is}_\text{tp} \alpha + \text{is}_\text{tm} x; x:A$

Lemma (Substitution)
If $\Phi_{\alpha t}, \text{is}_\text{tp} \alpha \vdash M : B$ and $\Phi_{\alpha t} \vdash \text{is}_\text{tp} A$, then $\Phi_{\alpha t} \vdash [A/\alpha]M : [A/\alpha]B$. 