A POPLmark retrospective

Using proof assistants in programming language research

Stephanie Weirich
University of Pennsylvania
The POPLmark Challenge

• A set of challenge problems meant to demonstrate the effectiveness of proof assistants in programming language research
• Issued at TPHOLs 2005
• Brian Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich and Steve Zdancewic
Why?

• A little PL research history...
• Since early 90s, trend in programming language research towards syntactic methods
A SYNTACTIC APPROACH TO TYPE SOUNDNESS

Andrew K. Wright*       Matthias Felleisen*

Department of Computer Science  
Rice University  
Houston, TX 77251-1892  

June 18, 1992

Rice Technical Report TR91-160  
To appear in: Information and Computation

Abstract

We present a new approach to proving type soundness for Hindley/Milner-style polymorphic type systems. The keys to our approach are (1) an adaptation of subject reduction theorems from combinatory logic to programming languages, and (2) the use of rewriting techniques for the specification of the language semantics. The approach easily extends from polymorphic functional languages to imperative languages that provide references, exceptions, continuations, and similar features. We illustrate the technique with a type soundness theorem for the core of STANDARD ML, which includes the first type soundness proof for polymorphic exceptions and continuations.
Define the syntax and type system of a simple language (STLC + unit)

\[
\begin{align*}
\tau & ::= \text{Unit} \\
& \quad | \tau_1 \to \tau_2 \\
\Gamma & ::= . \\
& \quad | \Gamma, x: \tau \\
\end{align*}
\]

\[
\begin{align*}
e & ::= \text{unit} \\
& \quad | x \\
& \quad | \lambda x: \tau. e \\
& \quad | e_1 e_2 \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash \text{unit} : \text{Unit} \quad \text{UNIT} \\
\end{align*}
\]

\[
\begin{align*}
\frac{(x: \tau) \in \Gamma}{\Gamma \vdash x : \tau} \quad \text{VAR} \\
\frac{\Gamma, x: \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x: \tau_1. e : \tau_1 \to \tau_2} \quad \text{ABS} \\
\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 \ e_2 : \tau_2} \quad \text{APP}
\end{align*}
\]
Now what is the semantics?

- **Denotational semantics**: maps programs to mathematical objects, such as functions.
- **Operational semantics**: describes how programs rewrite to values.

\[
\begin{align*}
\nu & ::= \text{unit} \quad \lambda x: \tau. e \\
\end{align*}
\]
Type soundness

'Well-typed programs don't go wrong':

Theorem 1 (Type Soundness) If \( \cdot \vdash e : \tau \) and \( e \) terminates, then there is some \( v \) such that \( e \rightarrow^* v \).

i.e. all evaluation either diverges or produces a valid final configuration.
### Syntactic Soundness Proof

**Lemma 2 (Preservation)** If \( \cdot \vdash e : \tau \) and \( e \rightarrow e' \) then \( \cdot \vdash e' : \tau \).

**Lemma 3 (Substitution)** If \( \Gamma, x:\tau' \vdash e : \tau \) and \( \Gamma \vdash e' : \tau' \) then \( \Gamma \vdash \{ e' / x \} e : \tau \).

**Lemma 4 (Progress)** If \( \cdot \vdash e : \tau \) and \( e \) is not a value, then there exists an \( e' \) such that \( e \rightarrow e' \).

**Lemma 5 (Canonical Forms)**

1. If \( \cdot \vdash v : \text{Unit} \) then \( v \) must be unit.

2. If \( \cdot \vdash v : \tau_1 \rightarrow \tau_2 \) then \( v \) must be \( \lambda x:\tau_1 . e \).

All of these lemmas proved by simple techniques (induction or inversion).
Why this technique?

• Low demands on the semanticists
  – Requires an 'operational' view of program execution
  – Easy to define because it resembles how the machine actually executes
  – Requires little mathematical machinery
    • "Just" inductive datatypes, alpha-conversion
• Proofs (of easy results) are easy
  – Series of straightforward inductions
  – Same form of lemmas each time
  – Cleverness is in setting up the type system the right way so that the usual properties work out
Covers many language features

references
transactional memory

nontermination
concurrency

polymorphism

exceptions

higher-order functions

dependent types
continuations

I/O

objects
aspects
A SYNTACTIC APPROACH TO TYPE SOUNDNESS

Andrew K. Wright*           Matthias Felleisen*

Department of Computer Science
Rice University
Houston, TX 77251-1892

June 18, 1992

Rice Technical Report TR91-160
To appear in: Information and Computation

Abstract

We present a new approach to proving type soundness for Hindley/Milner-style polymorphic type systems. The keys to our approach are (1) an adaptation of subject reduction theorems from combinatory logic to programming languages, and (2) the use of rewriting techniques for the specification of the language semantics. The approach easily extends from polymorphic functional languages to imperative languages that provide references, exceptions, continuations, and similar features. We illustrate the technique with a type soundness theorem for the core of STANDARD ML, which includes the first type soundness proof for polymorphic exceptions and continuations.
What is wrong with this method?

- Although the math is simple, there can be many cases.
- Syntactic methods mean intuition can fail.
- Easy to get something wrong in the details, especially in the combination of features.
The Standard ML of New Jersey implementation of callcc is not type safe, as the following counterexample illustrates:...

The counterexample does contradict a claim by Felleisen and Wright to the effect that the type system is sound; it is my understanding that they have repaired the proof by restricting the language.
The core of the type checking system was shown to be safe... but the type inference system for generic method calls was not subjected to formal proof. In fact, it is unsound ... This problem has been verified by the JSR14 committee, who are working on a revised language specification...
Again and again

From: Xavier Leroy
Sent: 30 Jul 2002
To: John Prevost
Cc: Caml-list
Subject: Re: [Caml-list] Serious typechecking error involving new polymorphism (crash)

... Yes, this is a serious bug with polymorphic methods and fields. Expect a 3.06 release as soon as it is fixed.

...
It happens to the best of us...

From: Dimitrios Vytiniotis
Subject: very serious bug in one lemma for completeness ...
Date: 21 April 2005
To: Stephanie Weirich
Cc: Simon Peyton Jones

As I was typing up the proofs I discovered that the strengthening lemma I have is not correct ... this might affect the whole paper ... Stephanie can we meet if you are around? (otherwise tomorrow ...) :-(( ... 

-d
Syntactic methods continue to be popular

- Foundation for programming language study
- But it *can* be too much of a good thing...
The State of the Art

Chen and Tarditi,
A Simple Typed Intermediate Language for Object-Oriented Languages,
Principles of Programming Languages (POPL), 2005

We have proved the soundness of $\text{LIL}_C$, in the style of [34], and the decidability of type checking. Full proofs are in the technical report.

**Theorem 1 (Preservation).** If $\Sigma \vdash P : \tau \text{ and } P \rightarrow P'$, then $\exists \Sigma'$ such that $\Sigma' \vdash P' : \tau$.

**Theorem 2 (Progress).** If $\Sigma \vdash P : \tau$, then either the main expression in $P$ is a value, or $\exists P'$ such that $P \rightarrow P'$.

Proof sketch: by standard induction over the typing rules.
The State of the Art

7/20/07

Proof by induction on expression typing rules.

Proof: By induction on expression typing rules.

Case int

Case record

Case array

Case if

Case typevar

Case class

Case where

Case forall

Case bind

Case assign

Case subst

Case not

Case let

Case e i

Case m i

Case alt

Case ternary

Case tag

Case calling

Case array

Case assign

Case subst

Case if

Case record

Case assign

Case subst

Case forall

Case bind

Case typevar

Case class

Case bind
## Personal Experience

<table>
<thead>
<tr>
<th>Publication</th>
<th>TR length</th>
<th>Heroic grad student</th>
</tr>
</thead>
<tbody>
<tr>
<td>JFP 07</td>
<td>83 pages</td>
<td>Dimitrios</td>
</tr>
<tr>
<td>ICFP 06</td>
<td>59 pages</td>
<td>Dimitrios</td>
</tr>
<tr>
<td>ICFP 06</td>
<td>58 pages</td>
<td>Dimitrios</td>
</tr>
<tr>
<td>ICFP 05</td>
<td>60 pages</td>
<td>Geoff, Dan</td>
</tr>
<tr>
<td>LICS 05</td>
<td>60 pages</td>
<td>Geoff</td>
</tr>
<tr>
<td>TLDI 04</td>
<td>51 pages</td>
<td>Geoff, Dimitrios</td>
</tr>
<tr>
<td>WOOD 04</td>
<td>49 pages</td>
<td>Liang</td>
</tr>
<tr>
<td>ICFP 03</td>
<td>61 pages</td>
<td>Geoff</td>
</tr>
</tbody>
</table>
Why write-only TRs?

Proofs optimized for conveying understanding

vs.

Proofs optimized for conveying certainty

i.e. we believe this is true because we actually worked out the details. And you can check our details if you have the patience.

Who has more patience than a machine?

Existing research community of logics for expressing such proofs and tools for checking them.
Some were already doing this...

- Leroy’s verified C compiler
- Nipkow et al’s formalization of a large part of Java
- Appel et al’s Foundational Proof-Carrying Code project
- Crary et al’s machine-checked development of a typed assembly language
- Harper et al’s formalization of Standard ML
- Sewell et al’s formalization of TCP/IP
- Etc., etc.
...but no common knowledge

- What proof assistant to use?
- How to get started? Manuals? Tutorials?
- Libraries?
- Existing developments?

The POPLmark challenge was a community and infrastructure building project
THE CHALLENGE, SPECIFICALLY

Metatheory of System F-sub
Challenge 1: Transitivity of subtyping

If $\Gamma \vdash S \leq Q$ and $\Gamma \vdash Q \leq T$, then $\Gamma \vdash S \leq T$.

- Transitivity must be proven *simultaneously* with narrowing, which states:

  \[
  \text{If } \Gamma, X \leq Q, \Gamma' \vdash S \leq T \text{ and } \Gamma \vdash P \leq Q, \text{ then } \Gamma, X \leq P, \Gamma' \vdash S \leq T.
  \]

- What's tested here: Non-trivial inductive proofs, isolating elements of the context
Challenge 2: Type safety

1. If $\Gamma \vdash e : T$ and $e \rightarrow e'$, then $\Gamma \vdash e' : T$.

2. If $\Gamma \vdash e : T$, then either $e$ is a value or else $e \rightarrow e'$ for some $e'$.

- Extended language with records and pattern matching
- What's tested here: Reasoning about syntax with variable numbers of components
  - Record patterns may bind arbitrarily many variables
  - Record values may contain an arbitrary number of fields
Challenge 3: "Animation"

1. Given $e$ and $e'$, decide if $e \rightarrow e'$.
2. Given $e$ and $e'$, decide if $e \rightarrow* e' \not\rightarrow$.
3. Given $e$, find $e'$ such that $e \rightarrow e'$.

- What's tested here: the ability to explore a language's properties on particular examples
- Solutions for (1) and (2) can check an interpreter
- Solution for (3) is an interpreter
Evaluation criteria

• Readers:
  – Adequacy of the encoding: Is it correct?
  – Obviousness of the encoding: How difficult is it to understand adequacy?

• Writers:
  – Clutter, inconvenience introduced by the technology
  – Effort required beyond a paper proof, even for experts

• Cost of entry:
  – Quality of documentation
  – Maturity of technology
What happened next?
POPLmark results

• Lots of interest!
• 15 submitted solutions recorded on wiki
  – 7 tools used (Coq, Isabelle/HOL, Twelf, ATS, Matita, Abella, Alpha-Prolog)
• Other solutions discussed elsewhere (ACL2, MetaPRL, Nominal-Isabelle)
"POPLmark tarpit"

• Techniques for representing variable binding caused the most heated discussion
  – 7 different techniques used in 15 solutions
  – Hit a pre-existing, active research area
• Our own efforts to understand this issue resulted in new research results
  – *Engineering Formal Metatheory, POPL 08*
    Aydemir, Chargueraud, Pierce, Pollack, Weirich
• Other parts of the challenge relatively ignored
  – Many did not complete full challenge with records or animation
Community development

• We worked hard to promote the use of proof assistants among PL researchers...
  – Organized workshops (4 instances of WMM so far)
  – Developed tutorial material
  – Developed a library for PL reasoning
  – Distributed all of our own developments
  – Integrated proof assistant use into our graduate PL course
Had to pick something...

- Devoted our efforts to Coq Proof Assistant
  - Wanted a general purpose logic
  - Wanted a mature platform
  - Constructive logic, dependent types were attractive

- Could have chosen others with equal success
  - Exciting new developments in the meantime: Nominal-Isabelle, Abella, etc.
It started to work...

- More papers with machine checked appendices start appearing
  - Some bootstrapped from our own work
- AURA – Zdancewic et al. ICFP 2007
  - Language for reasoning about authorization
  - Security-orientation motivates more certainty
  - Sophisticated dependent type system
  - Metatheory completely developed in Coq
  - 12.4k LOC
New tool - Ott: Sewell et al.

A tool should generate many outputs given a single “naturally written” definition of a language

- Language definition (ASCII)
- Infrastructure for mechanized proofs (Coq, Isabelle/HOL, Twelf)
- Infrastructure for writing papers/specs/talks (LaTeX)
- Implementation (Ocaml, Haskell)
Example: lambda terms

```
metavar atom, x, y, z ::= {{ coq nat }}{{ coq-equality }}

exp, e, f, g ::= '' ::= 
| x ::= var
| \ x . e ::= abs (+ bind x in e +)
| e1 e2 ::= app
| { e / x } e' ::= M ::= subst {{ coq subst [[e]] [[x]] [[e']] }}

substitutions
single e x ::= subst

defn
  e1 --> e2 ::= :: reduce ::'' by

  ----------------------------- :: ax_app
  (\x.e1) e2 --> {e2/x}e1

  e1 --> e1'
  -------------- :: ctx_app_fun
  e1 e --> e1' e

Ott code
```
Example: Typed lambda terms

Definition atom := nat.

Inductive exp : Set :=
| var : atom -> exp
| abs : atom -> typ -> exp -> exp
| app : exp -> exp -> exp.

Inductive reduce : exp -> exp -> Prop := (* defn reduce *)

| ax_app : forall (x:atom) (e12 e2:exp),
  reduce (app (abs x e12) e2) (subst e2 x e12 )

| ctx_app_fun : forall (e1 e_5 e1’:exp),
  reduce e1 e1’ ->
  reduce (app e1 e_5) (app e1’ e_5).
Lemma eq_atom: forall (x y : atom), \{x = y\} + \{x <> y\}.

... Fixpoint list_mem (A:Set) (eq:forall a b:A,\{a=b\}+\{a<>b\})
(x:A) (l:list A) \{struct l\} : bool := ...

Fixpoint subst (e_6:exp) (x5:atom) (e__7:exp) \{struct e__7\}
: exp :=
  match e__7 with
  | (var x) => (if eq_atom x x5 then e_6 else (var x))
  | (abs x e5) => abs x (if list_mem eq_atom x5 (cons x nil)
    then e5 else (subst e_6 x5 e5))
  | (app e5 t') => app (subst e_6 x5 e5) (subst e_6 x5 e')
  end.

Coq code output by OTT
How did the POPLmark challenge impact my research?
My research methods have changed

• I use OTT for all of my type setting
  – including parts of this talk
  – especially exploratory, development work
• I find formalizing the definitions in a paper often helps my understanding
• I sometimes pop open a Coq window to try out some thoughts
• Collaboration is easier this way
  – Version control
  – Definitions, proof status always up-to-date
• New research on variable binding
The issue with variable-binding

- Bound variables must alpha-vary
  - Identify $\ \backslash x.x$ and $\ \backslash y.y$

- Free variables must be 'sufficiently fresh'
  - Capture-avoiding substitution $e \{ e' / x \}$ --- bound variables in $e$ must not be the same as the free variables in $e'$
  - "Barandregt Variable Convention"
Locally nameless rep

• POPL 08 paper advocated two ideas for variable binding

  • Locally nameless representation (old idea)
    – Separate bound and free variables
    – Use numbers for bound variables (unique representation of alpha-equivalent terms) and strings for free variables

  • Cofinite quantification (new idea)
    – Premise of judgments quantifies over all variables except for some finite set
    – Strong induction principle
POPLmark challenge in Coq

Locally nameless definitions: OTT can generate these

Lemmas for substitution, weakening in judgments

Lemmas about free variable and substitution functions
Other experiences

• Rossberg, Russo, Dreyer. *F-ing modules.* TLDI 2010
• 13k line Coq development
• Used locally nameless approach
• 400 out of 550 lemmas were tedious "infrastructure" results
LNgen – Work in Progress

• Brian Aydemir and Stephanie Weirich. 
  *LNgen: Tool Support for Locally Nameless representations.*

• Works with OTT tool

• Generates and proves 'infrastructure' lemmas based on locally nameless representation

• Example lemma: if \( \text{fv}(t) = 0 \) then \([x \rightarrow u]t = t\)
Example: STLC development

- Ott (locally nameless backend) – 134 lines
  - 5 inductive definitions (typ, exp, lc, typing, step)
  - 3 functions (open, fv, subst)
  - 1 tactic (to collect all free vars in a proof)

- Lngen – 1533 lines
  - 3 functions (close, size_typ, size_exp)
  - 2 inductive definitions (degree, lc_set)
  - 47 lemmas
  - 2 tactics, 90 Hints

- Hand proofs – 108 lines
  - 8 lemmas (4 adequacy, weakening, subst, preservation, progress)
What are those 47 lemmas
Why proof generation is ok

• Code generators (rightly so) have a bad name
• Why is this a reasonable way to do things?
• Proof-irrelevance: don't care how a lemma was proved, only that it was proved
• lots of regular structure
  – F-omega: substitute types in terms, terms in terms, types in types
• Clear scope: Reasoning restricted to 5 operations
  – open, close, subst, fv, lc
  – lemmas concern only these operations and their interactions with each other
Case studies

• LNgen provided infrastructure for two POPL 2010 papers

• Greenberg, Weirich, Pierce. *Contracts Made Manifest*
  – Most proofs by hand (60 page TR)
  – Tricky reasoning about parallel reduction done in Coq.
    Replaced 8 dense pages of TR appendix

• Jia, Zhao, Sjöberg, Weirich. *Dependent Types and Program Equivalence*
  – Varied language for 9 months, doing proofs by hand
  – Used LNgen to check results in about 2 weeks
# Contracts

<table>
<thead>
<tr>
<th>Terms</th>
<th>Count</th>
<th>Generated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>terms.v</td>
<td>438</td>
<td>OTT</td>
</tr>
<tr>
<td>infrastructure.v</td>
<td>3965</td>
<td>LNgen</td>
</tr>
<tr>
<td>prelim.v</td>
<td>764</td>
<td></td>
</tr>
<tr>
<td>thy.v</td>
<td>3090</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>8257</td>
<td></td>
</tr>
</tbody>
</table>
## Dependent types

<table>
<thead>
<tr>
<th>File</th>
<th>Lines</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>lang.v</td>
<td>991</td>
<td>Generated by OTT</td>
</tr>
<tr>
<td>langExtra.v</td>
<td>267</td>
<td></td>
</tr>
<tr>
<td>infrastructure.v</td>
<td>7638</td>
<td>Generated by LNgen</td>
</tr>
<tr>
<td>isEq.v</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>thy.v</td>
<td>6116</td>
<td></td>
</tr>
<tr>
<td>thyPP.v</td>
<td>2126</td>
<td></td>
</tr>
<tr>
<td>progress.v</td>
<td>290</td>
<td></td>
</tr>
<tr>
<td>reductions.v</td>
<td>862</td>
<td></td>
</tr>
<tr>
<td>isEqSpecification.v</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>isEqBeta.v</td>
<td>691</td>
<td></td>
</tr>
<tr>
<td>isEqC.v</td>
<td>2284</td>
<td></td>
</tr>
<tr>
<td>inclusions.v</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>21592</td>
<td></td>
</tr>
</tbody>
</table>
## Proofs instead of TRs (mostly)

<table>
<thead>
<tr>
<th>Venue</th>
<th>Mech.</th>
<th>How</th>
<th>Tech report</th>
<th>Heroic students</th>
</tr>
</thead>
<tbody>
<tr>
<td>POPL 10</td>
<td>some</td>
<td>Coq</td>
<td>60 pages</td>
<td>Michael</td>
</tr>
<tr>
<td>POPL 10</td>
<td>yes</td>
<td>Coq</td>
<td></td>
<td>Limin (post-doc), Jianzhou, Vilhelm</td>
</tr>
<tr>
<td>PLPV 10</td>
<td>yes</td>
<td>Agda</td>
<td></td>
<td>Chris</td>
</tr>
<tr>
<td>CCS 09</td>
<td>yes</td>
<td>Coq</td>
<td></td>
<td>Aaron, Vilhelm</td>
</tr>
<tr>
<td>ICFP 08</td>
<td>no</td>
<td></td>
<td>Dissertation</td>
<td>Dimitrios</td>
</tr>
<tr>
<td>POPL 08</td>
<td>yes</td>
<td>Coq</td>
<td></td>
<td>Arthur, Brian</td>
</tr>
<tr>
<td>MFPS 07</td>
<td>yes</td>
<td>Isabelle/HOL</td>
<td></td>
<td>Dimitrios</td>
</tr>
</tbody>
</table>
Where to from here? What next for PL community?
Active research into variable binding

• Just in Cambridge:
  – Pitts – Nominal System T [POPL 2010]
  – Urban – Nominal Isabelle
  – Kennedy, Benton – Strongly typed Coq

• I don't think we have the complete story yet
Proof engineering

• Proof engineering
  – How to make sure that proofs are maintainable?
  – Haven't tactical theorem provers failed before?

• I don't know the answer to this problem
Role in Education

• Pierce: new textbook using Coq for grad students at Penn

• Excellent tool for teaching about proofs by induction, syntactic approach to programming language definitions, etc.

• What about discrete math?
Language definition

• What do we need to do to make sure that it is standard practice to have a machine-checked language specification?

• Again, heroic efforts exist...
  – SML, OCAML (light), Java (light)

• ... but consensus is necessary
  – Language designers want accessible specs
Hi, I have been looking on the web without success. Is there any paper/tech report that gives the precise rules of the pCIC as it is currently implemented in Coq 8.2. (something like a latex version of Chapter 4 from the reference manual)

There is a latex version of the reference manual in the Coq source archive and a pdf version at [http://coq.inria.fr/distrib/V8.2pl1/files/](http://coq.inria.fr/distrib/V8.2pl1/files/).

AFAIK there is no other description on paper of the entire set of features of pCIC in its 8.2 implementation. Note however that there is a work in progress by Gyesik Lee and Benjamin Werner on the set-theoretical model of a formulation of pCIC that is very close to Coq.
CONCLUSION
Conclusions

• I plan to keep on using proof assistants in my day to day research
The Success of Typed Languages

- It is difficult for programmers to prove properties about individual programs.
- Instead, language designers prove properties about languages that imply properties of all programs in that language.

Example: A scheme programmer must prove that his program never executes \((1 + \text{true})\).

An ML programmer knows this already.
Fundamental idea: Type safety

- Milner – Well typed programs don't go wrong
- i.e. programs maintain certain invariants during their execution
- those invariants are described by the type system
  - Functions called with particular forms of arguments
How to prove type safety?

• Since the early 90s, type safety proved 'syntactically'
• Two key lemmas:
  – Preservation: If a program type checks, and it takes a step, it will still type check
  – Progress: If a program type checks and it is not in an (approved) terminal configuration then it can take a step
Current state of the art: Ott

• Input: Language definitions in ASCII
  – Syntax (BNF grammar)
  – Binding specifications
  – Relations (Typing judgments, operational semantics)

• Output: multiple tool definitions
  – LaTeX: Typesetting macros
  – Proof assistants: Inductive datatypes; functions for free variables and substitution

• http://www.cl.cam.ac.uk/~pes20/ott/
What did we do?

- Compared submitted solutions with our own explorations:
  - FJ in Coq / Twelf / Isabelle/HOL
  - Parametricity theorem in Isabelle/HOL
  - Damas-Milner in Nominal-Isabelle
  - Created our own solutions to POPLmark challenge in Coq