TYPECHECKING PROOF SCRIPTS:

MAKING INTERACTIVE PROOF ASSISTANTS ROBUST

MARCEL GOH

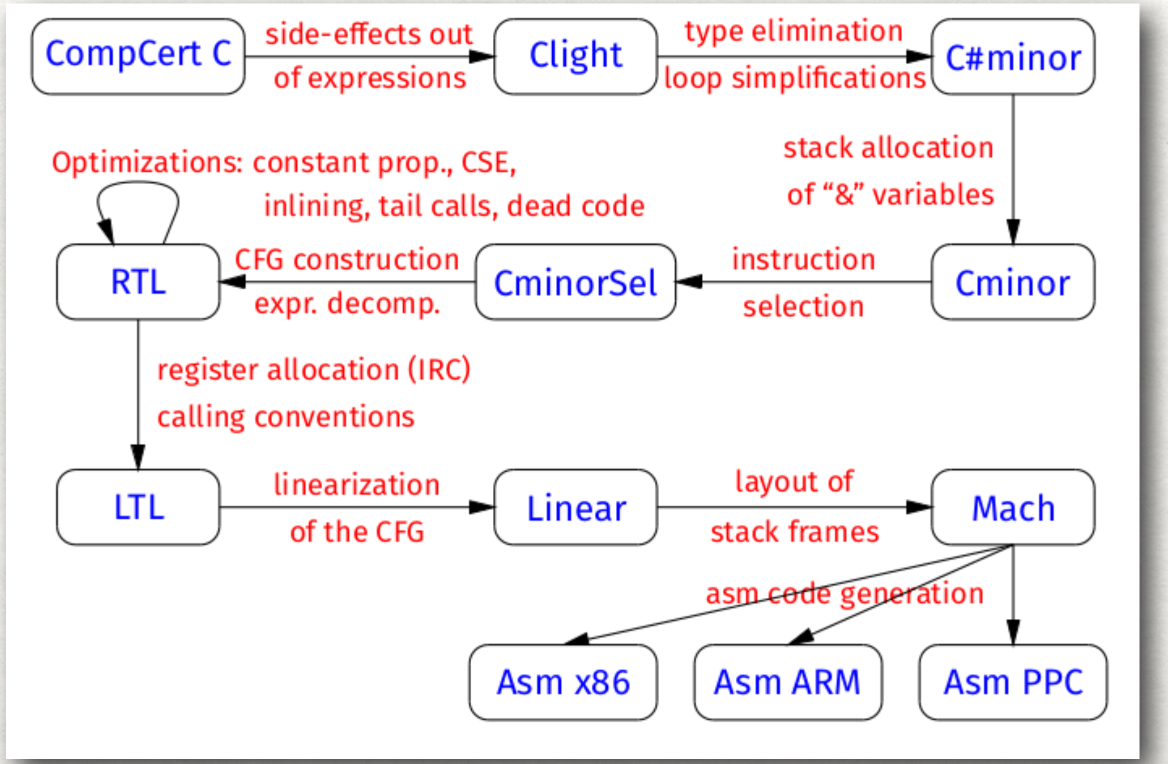
5 DECEMBER 2019

WHY PROVE THINGS ABOUT LANGUAGES?

- One motivation: nearly all software has bugs!
- Sometimes programs must be completely bulletproof (e.g. network security, avionics).
- Debugging: testing, static verification, etc.
- Better: prevent creation of bugs in the first place! (Formal models of languages.)
- Here proof assistants can come in handy.

BUG-FREE SOFTWARE: COMPCERT

- A compiler is just a program. It can be a weak link.
- CSmith (University of Utah): found 325+ bugs in GCC, Clang, and other popular C compilers.
- The only compiler found to have no bugs was CompCert, a C compiler written in Coq (X. Leroy, INRIA).
- Six CPU-years spent trying to find bugs in CompCert — none found, except in unproven parts (e.g. the parser).

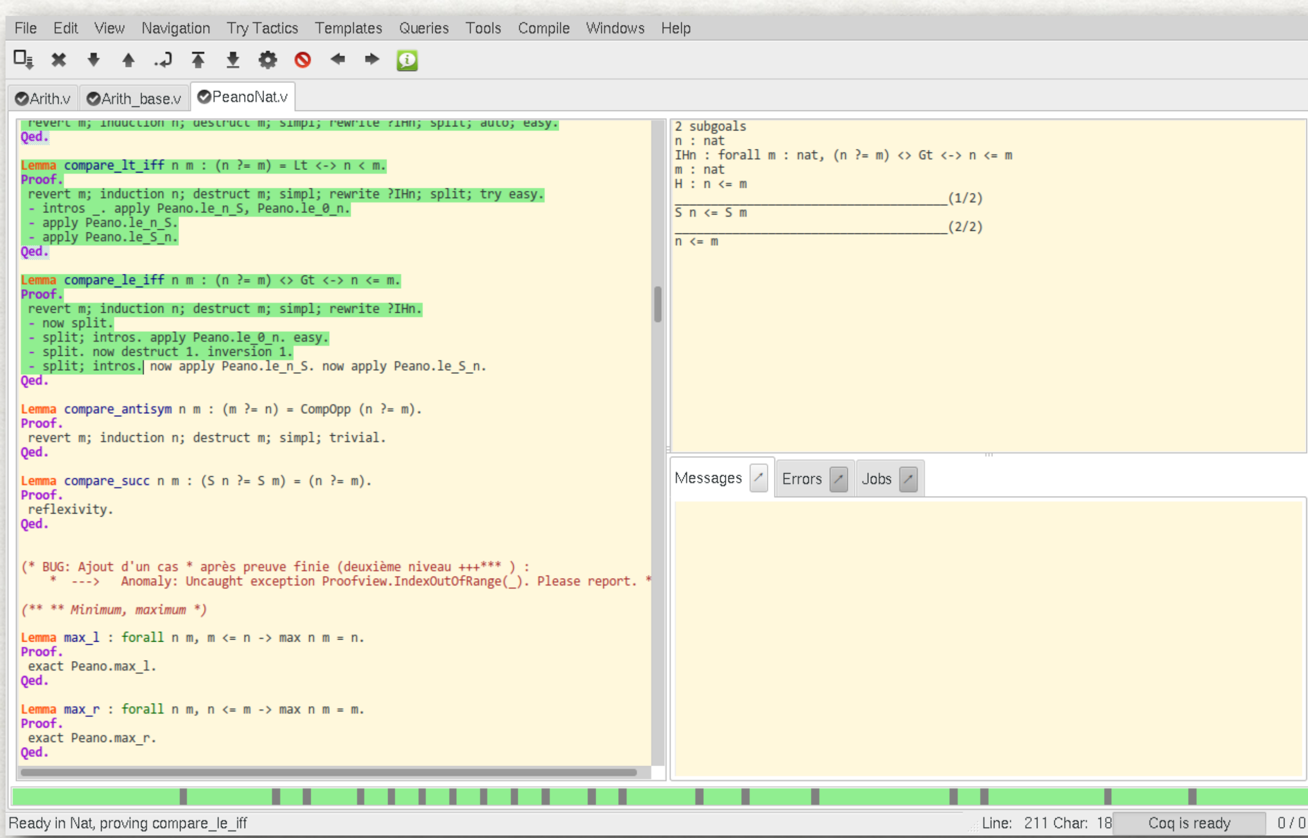


WHY TACTIC LANGUAGES?

Interactive construction of proofs: requires user guidance.

Happy medium: Tactics!

Fully automated proof search: Difficult (how to handle induction?)



Example: Coq

HARPOON: A TACTIC LANGUAGE FOR BELUGA

- Beluga is a functional programming language designed to reason about formal systems.
- Curry-Howard Correspondence: Beluga programs are proofs.
  - A function takes in arguments and returns an output.
  - A proof takes in hypotheses and returns a theorem.
  - Recursion = Induction
- Writing proofs by hand can be tricky and sometimes tedious.
- Harpoon: a tactic-based proof assistant for Beluga.

TACTICS IN HARPOON

- The Harpoon proof language is small, consisting of only a few tactics:
  - **intros**: Introduces the available assumptions.
  - **split**: Breaks an assumption up into its cases, generating new subgoals for each case.
  - **by lemma/by ih**: Invokes a previously-proven lemma, or invokes an induction hypothesis.
  - **unbox**: Converts a computation-level assumption into a meta-theoretic one.
  - **solve**: Once enough assumptions are present, prove the theorem.
- Harpoon includes facilities for solving trivial cases automatically.
- Output is a proof script, which can be checked and re-run.

MY CONTRIBUTIONS

- Designed typechecking rules for Harpoon proof scripts.
- Outlined a translation procedure from Harpoon proof scripts to Beluga programs.
- Proved the soundness of the translation procedure.
- **Theorem.** In contexts Δ and Γ, if a Harpoon proof script *P* checks against type τ and translates into Beluga term *t*, then the Beluga term *t* checks against type τ.
- Implementation in OCaml (in progress).

A SMALL PROOF: NATURAL NUMBERS

Axioms:

`le_z`: For all X, $0 \leq X$. `le_s`: If $X \leq Y$, then $\mathsf{succ}\, X \leq \mathsf{succ}\, Y$.

Theorem. *If $M \leq N$, then $M \leq \mathsf{succ}\, N$.*

Proof. We assume that $M \leq N$. Two ways we could have derived this:

i) From `le_s`.
There exist X, Y such that $M = \mathsf{succ}\, X$, $N = \mathsf{succ}\, Y$ and $X \leq Y$.
By induction, $X \leq Y$ means that $X \leq \mathsf{succ}\, Y$.
But $\mathsf{succ}\, Y = N$, so $X \leq N$.
We apply the axiom `le_s`: $X \leq N$ implies that $\mathsf{succ}\, X \leq \mathsf{succ}\, N$.
So $M \leq \mathsf{succ}\, N$.

ii) From `le_z`.
This means that $M = 0$.
We apply the axiom `le_z`:
$M \leq X$ for all X, so $M \leq \mathsf{succ}\, N$.

In both cases we proved that $M \leq \mathsf{succ}\, N$. ∎

THE HARPOON PROOF

Proof. We assume that $M \leq N$. Two ways we could have derived this:

i) From `le_s`.
There exist X, Y such that $M = \mathsf{succ}\, X$, $N = \mathsf{succ}\, Y$ and $X \leq Y$.
By induction, $X \leq Y$ means that $X \leq \mathsf{succ}\, Y$.
But $\mathsf{succ}\, Y = N$, so $X \leq N$.
We apply the axiom `le_s`: $X \leq N$ implies that $\mathsf{succ}\, X \leq \mathsf{succ}\, N$.
So $M \leq \mathsf{succ}\, N$.

ii) From `le_z`.
This means that $M = 0$.
We apply the axiom `le_z`:
$M \leq X$ for all X, so $M \leq \mathsf{succ}\, N$.

```
intros
{ {N : [ | - nat]}^i, {M : [ |- nat]}^i
| x1 : ([ |- leq M N])*
; meta-split (x1)
  case le_s:
  { {Z : [ |- leq X Z]}*, {X : [ |- nat]}*, {Y : [ |- nat]}*
  | x1* : ([ |- leq (succ Y) (succ X)])*
  ; by ih (lem [ |- Y] [ |- X] ([ |- Z])) as ih0;
    unbox (ih0) as IH0;
    solve ([ |- le_s IH0])
  }
  case le_z:
  { {N : [ |- nat]}*
  | x1* : ([ |- leq z N])*
  ; solve ([ |- le_z])
  }
```

HARPOON TO BELUGA

(Harpoon)

```
intros
{ {N : [ | - nat]}^i, {M : [ |- nat]}^i
| x1 : ([ |- leq M N])*
; meta-split (x1)
  case le_s:
  { {Z : [ |- leq X Z]}*, {X : [ |- nat]}*, {Y : [ |- nat]}*
  | x1* : ([ |- leq (succ Y) (succ X)])*
  ; by ih (lem [ |- Y] [ |- X] ([ |- Z])) as ih0;
    unbox (ih0) as IH0;
    solve ([ |- le_s IH0])
  }
  case le_z:
  { {N : [ |- nat]}*
  | x1* : ([ |- leq z N])*
  ; solve ([ |- le_z])
  }
```

```
rec lem : [ |- leq M N] -> [ |- leq M (succ N)] =
fn x1 =>
  case x1 of
  | [ |- le_s Z] =>
    let ih0 = lem [ |- Z] in
    let [ |- IH0] = ih0 in
    [ |- le_s IH0]
  | [ |- le_z] =>
    [ |- le_z]
;
```

(Beluga)

RECAP/CONCLUSION

- Formalising languages makes them more robust.
- Proof assistants help us prove things about languages.
- Curry/Howard: Proofs are programs!
- Alternate take: Proof assistants as an interactive medium for writing programs. (Always produce well-typed programs.)

PDF of slides available: https://marcelgoh.github.io/research

REFERENCES

- CompCert homepage: http://compcert.inria.fr/compcert-C.html
- Beluga homepage: http://complogic.cs.mcgill.ca/beluga/
- N. G. de Bruijn, "A Survey of the Project Automath," *Studies in Logic and the Foundations of Mathematics* **133** (1994), 141–161.
- X. Leroy, "Formal verification of a realistic compiler," *Communications of the ACM* **52** (2009), 107–115.
- X. Yang, Y. Chen, E. Eide, and J. Regehr, "Finding and Understanding Bugs in C Compilers,", *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '11)* (2011), 283–294.