How Logic and Philosophy of Logic Impact Computer Science

How to Program with Proofs

Aaron Stump

Dept. of Computer Science
The University of Iowa
Iowa City, Iowa, USA

Introductions

- Contacts in philosophy: Carrie Figdor, David Stern.
- U. Iowa Computational Logic Center:
  - Faculty: AS, Cesare Tinelli.
  - Postdocs: Garrin Kimmell, Tehme Kahsai.
  - Graduate students: Harley Eades III, Frank Fu, Tianyi Liang, Jed McClurg, Duckki Oe, Andy Reynolds, Cuong Thai.
  - Undergraduates: Austin Laugesen, JJ Meyer.
  - Funding: NSF, AFOSR (Air Force).
  - `http://clc.cs.uiowa.edu`
- Thank you for this invitation.

About This Talk

- Part 1: Proofs and Programs.
- Part 2: Connections with Philosophy.
- Main goals:
  - Show how proofs are computerized, and why.
  - Solicit your input on relevant ideas from Philosophy.

Proofs and Programs

What Use is Proof for Computer Science?

- Tremendous interest in computerized logical reasoning.
  - Need correctly functioning software and systems.
  - Deduction and proof provide universal guarantees.
  - **Verification**: prove system has specified properties.
- From this...

"seL4: formal verification of an OS kernel", Klein et al., SOSP 2009.

To this:

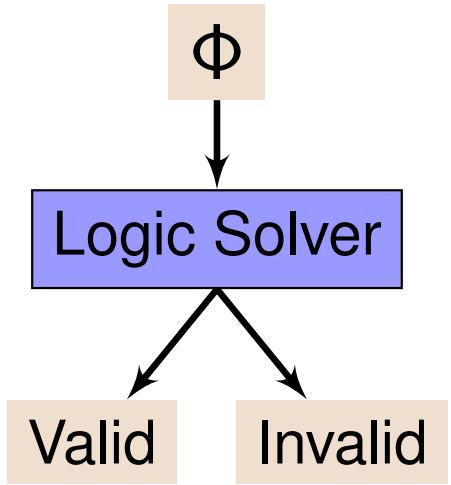
"Astrée: From Research to Industry", D. Delmas et al., SAS 2007.

Proofs and Size of Systems

- seL4 microkernel (mobile phones):
  - Around 9,000 lines of code.
  - 200,000 lines of computer-checked proof, written by hand.
  - Isabelle proof tool.
  - My estimate: 1 line of proof = 10 lines of code.
  - So equivalent to 2M lines of code.
- Airbus A380:
  - Millions of lines of code.
  - cf. Mercedes S-class: 100M lines of code.
  - Astrée can analyze 100Kloc programs.
- Why the difference in scale?

Two Kinds of Computer Proof

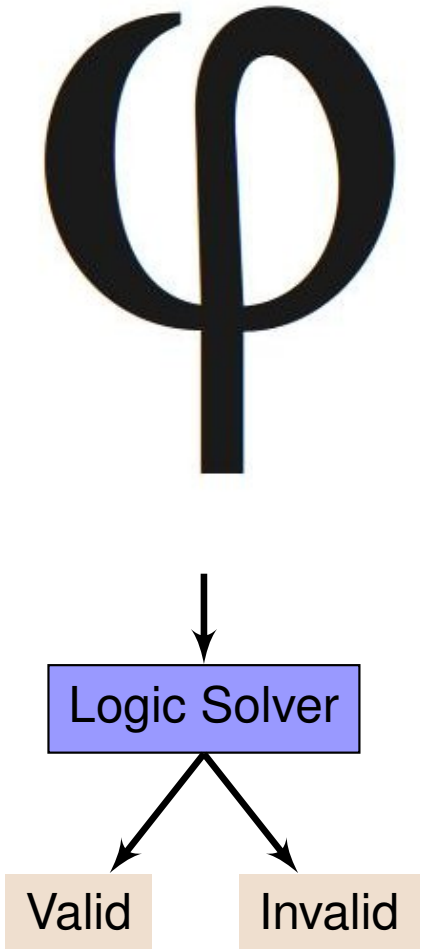
1. Automated Theorem Proving (Astrée).
   - Fully automatic.
   - Shallow reasoning, but
   - Large formulas.



2. Computer-Checked Manual Proof (Isabelle)
   - Written by hand.
   - Needed for deep reasoning.
   - Use solvers to fill in easy parts.

Large formulas (50 megabytes or more).

Large formulas (50 megabytes or more).



Programs as Proofs?

- Solvers test huge formulas.
- So solvers must be very efficient.
- So solvers must be complicated.
- What if the solver is wrong?
- *Quis custodiet custodes?*

Programs as Proofs?

- Solvers test huge formulas.
- So solvers must be very efficient.
- So solvers must be complicated.
- What if the solver is wrong?
- *Quis custodiet custodes?*

Haven't we seen this before?

Programs in Mathematical Proofs

- 1976. Appel and Haken prove Four Color Theorem.
  - Any map can be colored with at most 4 colors.
  - Open since 1800s.
  - Reduce problem to several thousand special cases.
  - Rule out those cases by computer program.
  - Infeasible for mathematicians to verify by hand.
  - Highly controversial!

Programs in Mathematical Proofs

- 1976. Appel and Haken prove Four Color Theorem.
  - Any map can be colored with at most 4 colors.
  - Open since 1800s.
  - Reduce problem to several thousand special cases.
  - Rule out those cases by computer program.
  - Infeasible for mathematicians to verify by hand.
  - Highly controversial!
- 2005. Gonthier gives computer-checked proof in Coq.
  - Coq certifies whole proof.
  - No trusted programs for special cases.
  - Gonthier proves those programs work as required.
  - Must only trust small core of Coq (around 6000 lines).

Another Example: Kepler Conjecture

- 1600s. Kepler conjecture on densest packing of spheres.
  - Conjecture: cannonball packing is densest.

Another Example: Kepler Conjecture

- 1600s. Kepler conjecture on densest packing of spheres.
  - Conjecture: cannonball packing is densest.
- 1998. Thomas Hales announces proof.
  - 250 pages, 3 gigabytes data and programs.
  - *Annals of Mathematics* gives to 12-member review panel.

Another Example: Kepler Conjecture

- 1600s. Kepler conjecture on densest packing of spheres.
  - Conjecture: cannonball packing is densest.
- 1998. Thomas Hales announces proof.
  - 250 pages, 3 gigabytes data and programs.
  - *Annals of Mathematics* gives to 12-member review panel.
- 2004. Review panel reports.
  - “99% certain proof is correct”.
  - But cannot certify correctness completely.
  - Editor writes: *“The news from the referees is bad, from my perspective. They have not been able to certify the correctness of the proof, and will not be able to certify it in the future, because they have run out of energy to devote to the problem. This is not what I had hoped for.”*

Another Example: Kepler Conjecture

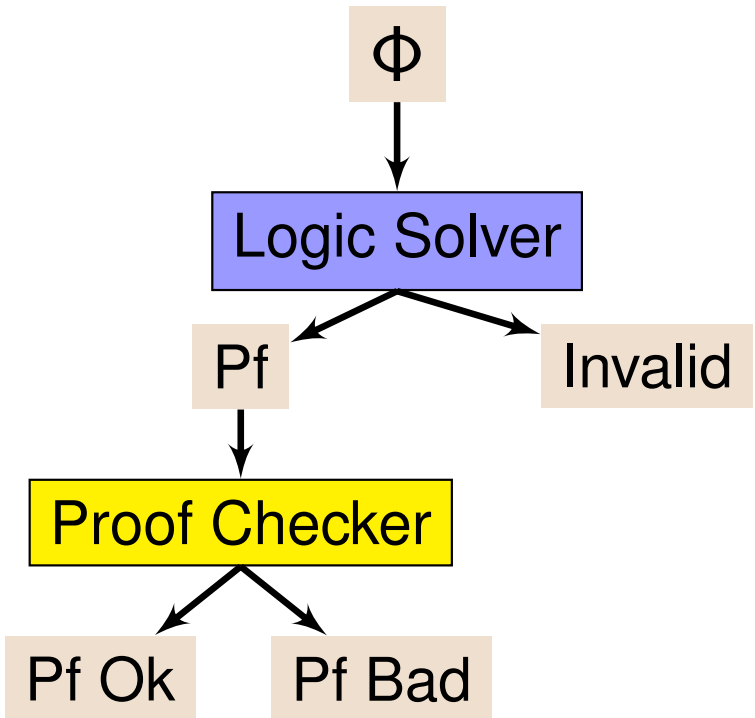
- 1600s. Kepler conjecture on densest packing of spheres.
  - Conjecture: cannonball packing is densest.
- 1998. Thomas Hales announces proof.
  - 250 pages, 3 gigabytes data and programs.
  - *Annals of Mathematics* gives to 12-member review panel.
- 2004. Review panel reports.
  - "99% certain proof is correct".
  - But cannot certify correctness completely.
  - Editor writes: *"The news from the referees is bad, from my perspective. They have not been able to certify the correctness of the proof, and will not be able to certify it in the future, because they have run out of energy to devote to the problem. This is not what I had hoped for."*
- 2004. Hales initiates `flyspeck` project.
  - Create a computer-checked version of his proof.
  - Estimated at 20 person-years of effort.
  - Google for latest status.

To Avoid Trusting a Solver:

1. Either prove solver correct.
   - Con: Requires deep reasoning, labor intensive.
   - Pro: Verified solver could be very valuable.
2. Or have solver produce proofs.
   - Proofs checked independently.
   - Proof checkers much simpler.

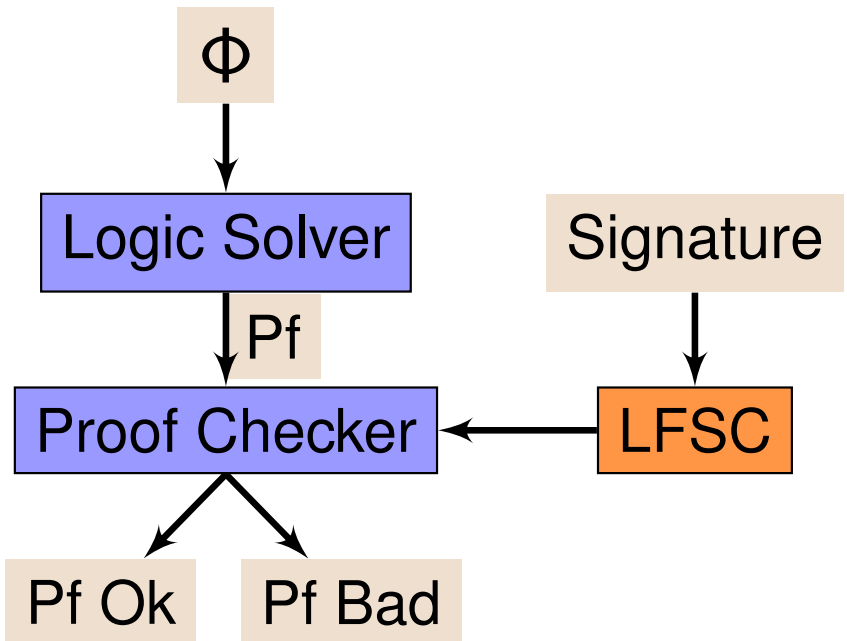
To Avoid Trusting a Solver:

1. Either prove solver correct.
   - Con: Requires deep reasoning, labor intensive.
   - Pro: Verified solver could be very valuable.
2. Or have solver produce proofs.
   - Proofs checked independently.
   - Proof checkers much simpler.



Producing Proofs from Solvers

- Big formulas => very big proofs (100MB to GBs).
- Need fast proof checker.
- Different solvers (dozens) => different proof systems.
- LFSC [McClurg,Reynolds,Stump,Thai,Tinelli].
  - "Logical Framework with Side Conditions".
  - Meta-language for describing logics.
  - Generate the proof checker from the logic description ("signature").



Benefits of LFSC

- Trustworthiness:
  - Declarative specification of proof checker.
  - Trusted: signature + generic LFSC compiler.
  - More trustworthy than hand-implemented checker.
  - More human-understandable.
- Flexibility:
  - Advanced solvers have hundreds of rules.
  - No consensus on single “right” proof system.
- Performance:
  - New optimizations implemented once in LFSC.
  - All proof systems can take advantage.

Example LFSC Signature: Natural Deduction

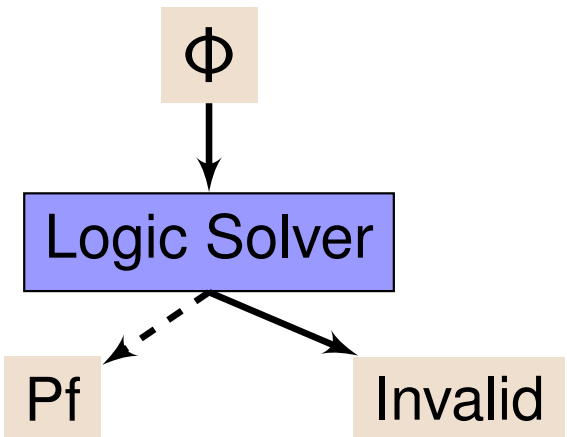
```
SYNTAX
individual i ::= Z | S i .
formula f ::= false | and f1 f2 | imp f1 f2 | forall i ^ f.

JUDGMENTS
(holds f)

RULES

holds f1 , holds f2
---------------------- and_intro
holds (and f1 f2) .

[ holds f1 ] |- holds f2
--------------------------  imp_intro
holds (imp f1 f2) .

i ^ holds (forall f{i})
---------------------- forall_elim
holds f{i} .
```

Proving a Solver Correct

`versat`: a verified modern SAT solver [Liang,**Oe**,Stump].

- Powerful propositional reasoning ("SAT") solvers.
- Can handle problems with 100K variables, 1M clauses.
- `versat` does not produce a proof.
- Instead, we prove:

If `versat` says "valid", there exists a proof.



Internal Verification and `versat`

- Could have:

```
Forall(F:formula)
       (u:(versat F) = valid).
Exists(p:proof F)
```

- `versat` is around 2000 lines of code.
- Very onerous to reason about externally.
- More feasible: internal verification.
  - Use types to state properties.

    ```
    versat : Fun(F:formula). answer F
    ```

  - The `answer` type is constructed by:

```
sat : Fun(spec F:formula). answer F
unsat : Fun(spec F:formula)(spec p:proof F). answer F
```

Programming With Proofs

- `versat` written in GURU.
  - Research language developed by AS, students.
  - Can prove theorems about programs.
  - Programs can also contain proofs.

    ```
    divide : Fun(x:int)(y:int)(u: y != 0). int
    ```

  - When `divide` is called, must supply a proof.

    ```
    divide 6 3 (clash 3 0)
    ```

  - Proofs are erased when program is run.
- Example of a combined language for proofs, programs.

Combined Languages for Programs and Proofs

- Many based on type theory.
  - Going back to Russell's theory of types.
  - Very active field, spanning Computer Science, Mathematical Logic.
  - Many proof assistants based on type theory.
- Many type theories use Curry-Howard (CH) isomorphism.
  - Based on striking similarity between proofs and programs.
    - ★ To prove $\phi(x)$ by induction, prove $\phi(0)$, then $\phi(n+1)$ assuming $\phi(n)$.
    - ★ To define $f(x)$ by recursion, define $f(0)$, then $f(n+1)$ using $f(n)$.
  - In CH: induction = recursion.
  - Advantage: unified syntax and semantics.
  - Disadvantage: not Turing-complete.
    - ★ Non-well-founded recursion = unsound induction.
    - ★ Must require all programs to terminate.
    - ★ Also, non-constructive reasoning not allowed.

Beyond Curry-Howard

- For practical use, need Turing-completeness.
- Might also like to use classical reasoning.
- So must abandon CH.
- Languages have two distinct parts: logical, programmatic.
- Several examples:
  - GURU.
  - “Logic-enriched type theory”, R. Adams, Z. Luo, 2010.
  - Our new language TRELLYS (U. Iowa, U. Penn. Portland State).

Vision: More correct software by programming with proofs

Connections with Philosophy

Constructivist Theories of Meaning

- Cf. "Logical Consequence from a Constructivist Point of View", Dag Prawitz, Oxford Handbook of Phil. of Math. and Logic, 2005.
- Meaning of logical connectives given by canonical forms of proof.
  - Canonical proof of $\phi_1 \wedge \phi_2$ is (p_1, p_2), where
  - p_1 is canonical proof of ϕ_1.
  - p_2 is canonical proof of ϕ_2.
- Introduction rules determine meaning:

$$\frac{\phi_1 \quad \phi_2}{\phi_1 \wedge \phi_2}\ \mathit{andI}$$

- Elimination rules for hypothetical reasoning.

$$\frac{\phi_1 \wedge \phi_2}{\phi_1}\ \mathit{andE}_1 \qquad \frac{\phi_1 \wedge \phi_2}{\phi_2}\ \mathit{andE}_2$$

Rules That Determine Meaning

- Guard against "tonk" example:

$$\frac{\phi_1}{\phi_1 \text{ tonk } \phi_2}\ \textit{tonkI} \qquad \frac{\phi_1 \text{ tonk } \phi_2}{\phi_2}\ \textit{tonkE}$$

- Require local soundness:

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \phi_1 & \phi_2\end{array}}{\phi_1 \wedge \phi_2}\ \textit{andI}}{\phi_1}\ \textit{andE}_1 \quad \Longrightarrow \quad \mathcal{D}_1$$

- Can also require local completeness:

$$\dfrac{\dfrac{\mathcal{D}}{\phi_1 \wedge \phi_2}\ \textit{andE}_1 \qquad \dfrac{\mathcal{D}}{\phi_1 \wedge \phi_2}\ \textit{andE}_2}{\phi_1 \wedge \phi_2}\ \textit{andI} \quad \Longrightarrow \quad \mathcal{D}$$

Semantic Confusion

- Constructivist account is for logical connectives only.
- Non-logical predicates may get meaning differently.
- What is a logical connective?
- CS theorists oversold on constructivist account.
- Try to define everything with intro/elim rules.
- Need (to explain to us) distinction between logical, non-logical.

Constructive vs. Classical Logic

- Type theory a hotbed of constructivism, predicativism.
- Curry-Howard => constructivist.
- For theories of programs, need classical reasoning.
  - E.g., case split on whether a program halts or not.
- Personally, anti-realist about mathematical truth.
- Independence results => alternate mathematical realities.
- Reject excluded middle, accept non-constructivity.

Foundations

- GURU, TRELLYS, others include new logics.
- Correctness of the language requires soundness of the logic.
- Unsoundness could be catastrophic:

```
fun(p : False). launch_warhead
```

- How to think about consistency proofs?
  - Gödel's 2nd Incompleteness Thm: need stronger theory.
  - (So cannot analyze "the strongest one".)
  - So what is gained by (computer-checked!) consistency proof?
- What are rational grounds to accept a foundational theory?

Future Connections

- Computer Scientists interested in reflective languages.
- From Philosophy, draw upon theories of reflective truth.
  - "Outline of a Theory of Truth", Saul Kripke, 1975.
  - Fixed-point approach to the truth predicate.
  - CS uses fixed-points for semantics all the time.
- Project idea: computer-checked theory of truth.
  - Pick a modern variant of Kripke's theory.
  - Either formalize directly in tool like Coq,
  - Or else devise type-theoretic version.
    - ★ Encode self-referential formulas coinductively?
      (*) "(*) is false" (*) = not (*) not (not (not (not ...)))
    - ★ Treat self-referential languages similarly?

$$L = L + T(\langle L \rangle)$$

    - ★ Define semantics use Kripke's idea?
  - Could shed light on reflection and formalized semantics.

Conclusion

- Surveyed connections between programming and proofs.
  - Proofs needed for system correctness.
  - Solvers for fast, shallow reasoning.
  - Computer-checked human proof for deep reasoning.
  - Solvers can produce proofs, or be proved correct.
  - Languages like GURU, TRELLYS combine programming, proving.
- Some philosophical connections:
  - How to think about semantics (logical/non-logical distinction)?
  - How to think about foundations?
  - Computer-checked theory of truth?
- Slides and cited papers online at my blog, QA9:

`queuea9.wordpress.com`

Thank you again!