Programs, Proofs, and Classical Logic

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

About This Talk

Part 1:	The Verification Renaissance
Part 2:	`versat`, a Verified Modern SAT Solver
Part 3:	Classical Proofs as Programs
Ad:	*U. Iowa CS grad programs*

Verification Reborn

Language-Based Verification Will Change the World,
T. Sheard, A. Stump, S. Weirich, FoSER 2010.

Computing systems are doing so much:

Why can’t we guarantee they work?

Computing systems are doing so much:

Why can’t we guarantee they work?

Why not just use testing?

- + Integrates well with programming
- + No new languages, tools required
- + Conclusive evidence for bugs

Why not just use testing?

\+ Integrates well with programming
\+ No new languages, tools required
\+ Conclusive evidence for bugs

– Difficult to assess coverage
– Cannot demonstrate absence of bugs
– No guarantees for safety-critical systems

Alternative: Formal Verification

Instead of tests, use proofs

- Deduction and proof provide universal guarantees
- Prove that software 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

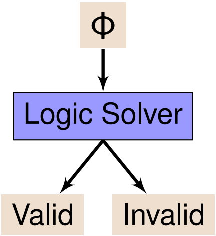
Proofs and Size of Systems

- seL4 microkernel (mobile phones):
  - Around 9,000 lines of code
  - 200,000 lines of computer-checked proof, written by hand
  - Isabelle proof tool
- 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

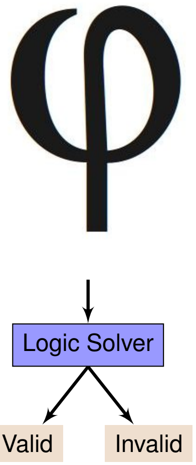
1. Automated Theorem Proving (Astrée)
   - Fully automatic
   - Shallow reasoning, but
   - Large formulas



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

Large formulas: megabytes

Large formulas: megabytes



Programs as Proofs?

- Solvers test huge formulas
  - $\Rightarrow$ solvers must be very efficient
  - $\Rightarrow$ solvers must be complicated
- What if the solver is wrong?
- Who watches the watchers?

About This Talk

Part 1:	The Verification Renaissance
Part 2:	`versat`, a Verified Modern SAT Solver
Part 3:	Classical Proofs as Programs
Ad:	*U. Iowa CS grad programs*

`versat`

A Verified Modern SAT Solver

versat: A Verified Modern SAT Solver,
D. Oe, A. Stump, C. Oliver, K. Clancy, VMCAI 2012.

SAT: Propositional Satisfiability

- Given a propositional formula, test *satisfiability*
  - Propositional: and ($\wedge$), or ($\vee$), not ($\neg$), variables (p, q, r)
  - Satisfiable: boolean values for variables exist making formula true
- Examples:
  - Satisfiable: $p \wedge (q \vee \neg p)$
    Set $p = \texttt{true}$ and $q = \texttt{true}$
  - Unsatisfiable: $(p \rightarrow q) \wedge (q \rightarrow r) \wedge p \wedge \neg r$
- Many optimizations for SAT solvers in last 15 years
  - Solvers can handle huge formulas (100k vars, 1M clauses)
- Validating answers:
  - When satisfiable, can check assignment
  - When unsatisfiable, some solvers dump proofs

`versat` Overview

- SAT solver with modern optimizations
- Implemented in GURU
  - Research language developed in my group
  - Used for verified programming
  - Combine rich types with inductive proofs
- Statically verified `unsat`-soundness
  - If `versat` says `unsat`
  - Then input formula is contradictory
- `sat`-soundness not verified
- Efficiency:
  - Uses standard efficient data structures
  - Can handle formulas on modern scale
- Around 2kloc code, 8kloc proofs

Main Specification

- The `solve` function has type:

```
Fun(F:formula)(...).<answer F>
```

- `answer` records proof for `unsat` case:

```
Inductive answer : Fun(F:formula).type :=
  sat  : Fun(spec F:formula).<answer F>
| unsat : Fun(spec F:formula)(spec p:<pf F False>).
          <answer F>
```

- `pf` is an indexed datatype of propositional proofs
- Data of type `<pf F1 F2>` are proofs that `F1` entails `F2`
- We have proved that a propositional proof exists
- Not constructed at run-time
  - Some solvers actually emit such proofs
  - Requires much extra time, space

Results: `versat` vs. proof checking

The Certified Track benchmarks of SAT Competition 2007

- 16 benchmarks (believed to be UNSAT)
- System: Intel Core 2 Duo 2.40GHz w/ 3GB of memory
- One hour timeout for solving and checking, individually

Systems	#Solved	#Certified
versat	6	6
picosat + RUP	14	4
picosat + TraceCheck	14	12

Trusted Base:

- `versat`: GURU compiler + 259 lines of GURU code
- checker3 (RUP checker): 1,538 lines of C code
- tracecheck (TraceCheck checker): 2,989 lines of C code + `boolforce` library (minisat-2.2.0 is ≈2,500 lines of C++)

The Broken-Window Theory

- Fix broken windows and the neighborhood improves
- Verify some properties and others hold, too

$\Longrightarrow$

- For `versat`:
  - We proved unsat-soundness
  - What about sat-soundness?

Fuzzing `versat`

- `cnfuzz` generates random instances [Brummayer+2010]
- Used to find bugs in competition SAT solvers (2007, 2009)
- Applied to `versat`:
  - Generated 10,000 random formulas
  - 54% satisfiable
  - Compare `versat` and MiniSat 2.2.0, 60 second timeout
  - `versat` timed out on 27 formulas
  - Otherwise, complete agreement

Fuzzing `versat`

- `cnfuzz` generates random instances [Brummayer+2010]
- Used to find bugs in competition SAT solvers (2007, 2009)
- Applied to `versat`:
  - Generated 10,000 random formulas
  - 54% satisfiable
  - Compare `versat` and MiniSat 2.2.0, 60 second timeout
  - `versat` timed out on 27 formulas
  - Otherwise, complete agreement

Wow!

About This Talk

Part 1:	The Verification Renaissance
Part 2:	`versat`, a Verified Modern SAT Solver
Part 3:	Classical Proofs as Programs
Ad:	*U. Iowa CS grad programs*

Classical Proofs as Programs

A Logic Game

Them	Us
	Defend: $A \to (B \to A)$

A Logic Game

Them	Us
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	

A Logic Game

Them	Us
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A	Defend: $B \to A$

A Logic Game

Them	Us
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A	Defend: $B \to A$
Attacking $B \to A$	

A Logic Game

Them	Us
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A	Defend: $B \to A$
Attacking $B \to A$	
Grant: B	Defend: A

A Logic Game

Them	Us
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A	Defend: $B \to A$
Attacking $B \to A$	
Grant: B	Defend: A
	Win (A granted).

The Rules of the Game

- To attack $A \rightarrow B$, must grant (defend) A and attack B
- To attack $A \wedge B$, can attack A or attack B
- To attack $A \vee B$, opponent can defend A or defend B
- If no move, other player continues
- First player wins if defending formula granted by other player

Another Example

Them	Us
	Defend: $(A \wedge B) \rightarrow (A \vee B)$

Another Example

Them	Us
	Defend: $(A \wedge B) \to (A \vee B)$
Attacking $(A \wedge B) \to (A \vee B)$	

Another Example

Them	Us
	Defend: $(A \wedge B) \rightarrow (A \vee B)$
Attacking $(A \wedge B) \rightarrow (A \vee B)$	
Grant: $A \wedge B$	Defend: $A \vee B$

Another Example

Them	Us
	Defend: $(A \wedge B) \rightarrow (A \vee B)$
Attacking $(A \wedge B) \rightarrow (A \vee B)$	
Grant: $A \wedge B$	Defend: $A \vee B$
	Attacking $A \wedge B$

Another Example

Them	Us
	Defend: $(A \wedge B) \rightarrow (A \vee B)$
Attacking $(A \wedge B) \rightarrow (A \vee B)$	
Grant: $A \wedge B$	Defend: $A \vee B$
	Attacking $A \wedge B$
	Attacking A

Another Example

Them	Us
	Defend: $(A \wedge B) \rightarrow (A \vee B)$
Attacking $(A \wedge B) \rightarrow (A \vee B)$	
Grant: $A \wedge B$	Defend: $A \vee B$
	Attacking $A \wedge B$
	Attacking A
Defend: A	

Another Example

Them	Us
	Defend: $(A \wedge B) \rightarrow (A \vee B)$
Attacking $(A \wedge B) \rightarrow (A \vee B)$	
Grant: $A \wedge B$	Defend: $A \vee B$
	Attacking $A \wedge B$
	Attacking A
Defend: A	
Attack: $A \vee B$	

Another Example

Them	Us
	Defend: $(A \wedge B) \rightarrow (A \vee B)$
Attacking $(A \wedge B) \rightarrow (A \vee B)$	
Grant: $A \wedge B$	Defend: $A \vee B$
	Attacking $A \wedge B$
	Attacking A
Defend: A	
Attack: $A \vee B$	
	Defend: A

Another Example

Them	Us
	Defend: $(A \wedge B) \rightarrow (A \vee B)$
Attacking $(A \wedge B) \rightarrow (A \vee B)$	
Grant: $A \wedge B$	Defend: $A \vee B$
	Attacking $A \wedge B$
	Attacking A
Defend: A	
Attack: $A \vee B$	
	Defend: A
	Win (A granted)

Constructive Logic

- Considering **constructive** (aka, intuitionistic) logic
  - To prove $A \vee B$, must prove one or the other
  - To prove $\exists x.A$, must have a *witness* for x
- Example constructive proof:

Theorem

There exist irrational numbers x and y such that x^y is rational.

Proof. Take $\sqrt{2}$ for x and $\log_2 9$ for y:

$$\sqrt{2}^{\log_2 9} = 2^{\frac{\log_2 9}{2}} = 2^{\log_2(9^{\frac{1}{2}})} = 9^{\frac{1}{2}} = 3$$

QED

Constructive Proofs and Programs

You	Me
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A	Defend: $B \to A$
Attacking $B \to A$	
Grant: B	Defend: A
	Win (A granted)

Constructive Proofs and Programs

You	Me
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A $\boxed{\lambda x.}$	Defend: $B \to A$
Attacking $B \to A$	
Grant: B	Defend: A
	Win (A granted)

Constructive Proofs and Programs

You	Me
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A $\boxed{\lambda x.}$	Defend: $B \to A$
Attacking $B \to A$	
Grant: B $\boxed{\lambda y.}$	Defend: A
	Win (A granted)

Constructive Proofs and Programs

You	Me
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A $\boxed{\lambda x.}$	Defend: $B \to A$
Attacking $B \to A$	
Grant: B $\boxed{\lambda y.}$	Defend: A
	$\boxed{x}$ Win (A granted)

Constructive Proofs and Programs

You	Me
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A $\boxed{\lambda x.}$	Defend: $B \to A$
Attacking $B \to A$	
Grant: B $\boxed{\lambda y.}$	Defend: A
	$\boxed{x}$ Win (A granted)

The program is $\lambda x.\lambda y.x$.

Constructive Proofs and Programs

You	Me
	Defend: $A \to (B \to A)$
Attacking $A \to (B \to A)$	
Grant: A $\boxed{\lambda x.}$	Defend: $B \to A$
Attacking $B \to A$	
Grant: B $\boxed{\lambda y.}$	Defend: A
	$\boxed{x}$ Win (A granted)

The program is $\lambda x.\lambda y.x$.

So *lambda calculus* can serve as language of proofs.

Non-Constructive Reasoning

- Non-constructive: $A \vee \neg A$, also $(\neg\neg A) \to A$
  - Suppose A is "Turing-machine M halts"
  - Constructive proof of $A \vee \neg A$ solves the halting problem for M
  - (So this is impossible)
- Example non-constructive proof:

Theorem

There exist irrational numbers x and y such that x^y is rational.

Proof. Case split on whether or not $\sqrt{2}^{\sqrt{2}}$ is rational.
If so, done. If not:

$$(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$$

QED

Logic Game for Non-Constructive Formulas

Let $\perp$ denote falsity.

Them	Us
	Defend: $((A \rightarrow \perp) \rightarrow \perp) \rightarrow A$

Logic Game for Non-Constructive Formulas

Let $\perp$ denote falsity.

Them	Us
	Defend: $((A \to \perp) \to \perp) \to A$
Attacking $((A \to \perp) \to \perp) \to A$	

Logic Game for Non-Constructive Formulas

Let $\perp$ denote falsity.

Them	Us
	Defend: $((A \rightarrow \perp) \rightarrow \perp) \rightarrow A$
Attacking $((A \rightarrow \perp) \rightarrow \perp) \rightarrow A$	
Grant: $(A \rightarrow \perp) \rightarrow \perp$	Defend: A

Logic Game for Non-Constructive Formulas

Let $\perp$ denote falsity.

Them	Us
	Defend: $((A \to \perp) \to \perp) \to A$
Attacking $((A \to \perp) \to \perp) \to A$	
Grant: $(A \to \perp) \to \perp$	Defend: A
	Attacking $(A \to \perp) \to \perp$

Logic Game for Non-Constructive Formulas

Let $\perp$ denote falsity.

Them	Us
	Defend: $((A \to \perp) \to \perp) \to A$
Attacking $((A \to \perp) \to \perp) \to A$	
Grant: $(A \to \perp) \to \perp$	Defend: A
	Attacking $(A \to \perp) \to \perp$
Defend: $\perp$	Grant: $A \to \perp$

Logic Game for Non-Constructive Formulas

Let $\perp$ denote falsity.

Them	Us
	Defend: $((A \to \perp) \to \perp) \to A$
Attacking $((A \to \perp) \to \perp) \to A$	
Grant: $(A \to \perp) \to \perp$	Defend: A
	Attacking $(A \to \perp) \to \perp$
Defend: $\perp$	Grant: $A \to \perp$
Attacking $A \to \perp$	

Logic Game for Non-Constructive Formulas

Let $\perp$ denote falsity.

Them	Us
	Defend: $((A \to \perp) \to \perp) \to A$
Attacking $((A \to \perp) \to \perp) \to A$	
Grant: $(A \to \perp) \to \perp$	Defend: A
	Attacking $(A \to \perp) \to \perp$
Defend: $\perp$	Grant: $A \to \perp$
Attacking $A \to \perp$	
Grant: A	Defend: $\perp$

Logic Game for Non-Constructive Formulas

Let $\bot$ denote falsity.

Them	Us
	Defend: $((A \rightarrow \bot) \rightarrow \bot) \rightarrow A$
Attacking $((A \rightarrow \bot) \rightarrow \bot) \rightarrow A$	
Grant: $(A \rightarrow \bot) \rightarrow \bot$	Defend: A
	Attacking $(A \rightarrow \bot) \rightarrow \bot$
Defend: $\bot$	Grant: $A \rightarrow \bot$
Attacking $A \rightarrow \bot$	
Grant: A	Defend: $\bot$
	Win (A granted).

Logic Game for Non-Constructive Formulas

Let $\perp$ denote falsity.

Them	Us
	Defend: $((A \rightarrow \perp) \rightarrow \perp) \rightarrow A$
Attacking $((A \rightarrow \perp) \rightarrow \perp) \rightarrow A$	
Grant: $(A \rightarrow \perp) \rightarrow \perp$	Defend: A
	Attacking $(A \rightarrow \perp) \rightarrow \perp$
Defend: $\perp$	Grant: $A \rightarrow \perp$
Attacking $A \rightarrow \perp$	
Grant: A	Defend: $\perp$
	Win (A granted).

We won by satisfying an earlier obligation!

Another Non-Constructive Game

$\neg A$ abbreviates $A \rightarrow \perp$.

Them	Us
	Defend: $A \vee \neg A$

Another Non-Constructive Game

$\neg A$ abbreviates $A \to \bot$.

Them	Us
	Defend: $A \vee \neg A$
Attacking $A \vee \neg A$	

Another Non-Constructive Game

$\neg A$ abbreviates $A \rightarrow \perp$.

Them	Us
	Defend: $A \vee \neg A$
Attacking $A \vee \neg A$	
	Defend: A

Another Non-Constructive Game

$\neg A$ abbreviates $A \rightarrow \perp$.

Them	Us
	Defend: $A \vee \neg A$
Attacking $A \vee \neg A$	
	Defend: A
	Defend: $\neg A$

Another Non-Constructive Game

$\neg A$ abbreviates $A \rightarrow \perp$.

Them	Us
	Defend: $A \vee \neg A$
Attacking $A \vee \neg A$	
	Defend: A
	Defend: $\neg A$
Attacking $A \rightarrow \perp$	

Another Non-Constructive Game

$\neg A$ abbreviates $A \to \bot$.

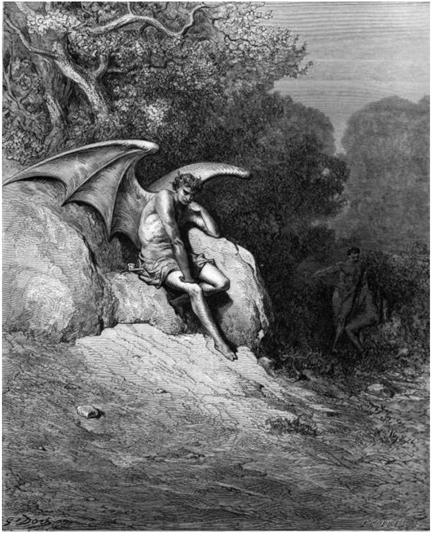
Them	Us
	Defend: $A \vee \neg A$
Attacking $A \vee \neg A$	
	Defend: A
	Defend: $\neg A$
Attacking $A \to \bot$	
Grant: A	Defend: $\bot$

Another Non-Constructive Game

$\neg A$ abbreviates $A \to \perp$.

Them	Us
	Defend: $A \vee \neg A$
Attacking $A \vee \neg A$	
	Defend: A
	Defend: $\neg A$
Attacking $A \to \perp$	
Grant: A	Defend: $\perp$
	Win: A granted

$A \vee (A \rightarrow \bot)$: a Devil's Bargain (as told by Phil Wadler)



Going Back In Time?

- Classical reasoning: win by satisfying earlier obligations
- Like going back to an earlier state
- Q. What programming feature is like that?
- A. Exceptions!
- For $((A \to \bot) \to \bot) \to A$:

$$\lambda f.catch\ y.(f\ (\lambda x.throw\ x\ to\ y))$$

  - $f :^{+} (A \to \bot) \to \bot$
  - $x :^{+} A$
  - $y :^{-} A$
  - $throw\ x\ to\ y :^{+} \bot$

Back to the Future?

- Exceptions useful for programming
- Also useful for classical proofs!
- Extensions of lambda calculus with exceptions
  - Duality between input/output
  - Use for duality between inductive/coinductive types
  - Extend programs-as-proofs to classical logic
- May serve as foundation for next generation of proof tools

Conclusion

- Verification: prove properties of programs
- Case study: `versat`
  - First verification of efficient modern SAT solver
- Non-constructive proofs as programs raising exceptions
  - New foundations for program-verification tools?
- Slides online at my blog, QA9:

`http://queuea9.wordpress.com`

Graduate Study in CS at U. Iowa

U. Iowa CS Grad Programs

- MCS: Master of Computer Science
  - Course-based program
  - Deepen CS knowledge beyond undergraduate curriculum
  - Basically: 10 CS graduate courses
  - No guarantees, but many TAships available
  - Strengthen credentials for industry
- PhD: Doctor of Philosophy
  - Research-based program
  - Develop students into independent researchers
    - Building systems, designing algorithms, proving theorems, etc.
  - Minimal, flexible course requirements
  - Funding through RAships, TAships, fellowships
  - Leads to careers in academics, research labs, and industry

Research Areas

- Algorithms (Pemmaraju, Varadarajan)
  - Computational Geometry, Approximation and Randomization
- Computational Logic (Stump, Tinelli, Zhang)
  - Verification, Programming Languages, Automated Theorem Proving
- Graphics, HCI (Cremer, Hourcade, Kearney, Wyman)
  - Interactive Rendering, Virtual Environments, Assistive Technologies
- Informatics (Segre, Srinivasan)
  - Text/Web Mining, Social Network Analysis, Comp. Epidemiology
- Distributed Systems (Chipara, Gosh, Herman)
  - Sensor Networks, Fault Tolerance, Distributed Algorithms
- Numerical Methods (Oliveira)
- Voting Technology (Jones)

Yelena Majova, PhD 2012 (Advisor: Srinivasan)

"My PhD research was on opinion extraction and sentiment analysis of social media text."

"I am a post-doc at Yahoo! Research in Barcelona, working on semantic entity, relationship, and property extraction from free text and image collections, serendipitous search, user behavior tracking using web data, and continuing political speech analysis."

Duckki Oe, PhD 2012 (Advisor: Stump)

Started postdoc at MIT August 2012.

What are your career plans after MIT?

“I’d certainly want to research and apply formal verification methods. My first choice would be a faculty position (preferably in Korea).

How do you feel about your time at Iowa CS?

“I feel very grateful that I had a very supportive advisor (mentally and financially) and other professors who are respected in my field. [...] Studying and raising children at the same time wasn’t easy. But, it was easier at Iowa because of child-friendly environment, low child care expenses and subsidy from the school.”

Thanks for listening!