COMP4011/8011
Advanced Topics in
Formal Methods and Programming Languages

– Software Verification with Isabelle/HOL –

Peter Höfner

September 22, 2024

Section 14

Sledgehammer and Co.

Overview

Automatic Proof and Disproof

- Sledgehammer: automatic proofs
- Quickcheck: counter example by testing
- Nitpick: counter example by SAT

Based on slides by Jasmin Blanchette, Lukas Bulwahn, and Tobias Nipkow (TUM).

Automation

Dramatic improvements in fully automated proofs in the last 2 decades.

- First-order logic (ATP): Otter, Vampire, E, SPASS
- Propositional logic (SAT): MiniSAT, Chaff, RSat
- SAT modulo theory (SMT): CVC3/4/5, Yices, Z3

The key:

Efficient reasoning engines, and restricted logics.

Automation in Isabelle

1980s *rule applications, write ML code*

1990s *simplifier, automatic provers (blast, auto), arithmetic*

2000s *embrace external tools, but don’t trust them (ATP/SMT/SAT)*

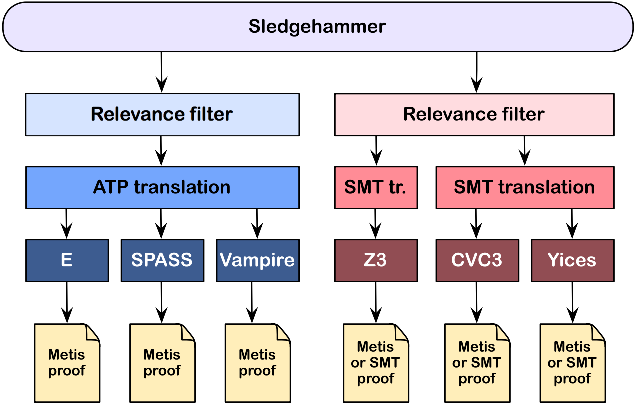
Sledgehammer

Sledgehammer:

- *Connects Isabelle with ATPs and SMT solvers:*
  E, SPASS, Vampire, CVC4, Yices, Z3
- *Simple invocation:*
  - *Users don’t need to select or know facts*
  - *or ensure the problem is first-order*
  - *or know anything about the automated prover*
- *Exploits local parallelism and remote servers*

Demo: Sledgehammer

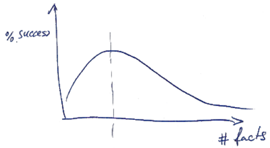
Sledgehammer Architecture



Fact Selection

Provers perform poorly if given 1000s of facts.

- *Best number of facts depends on the prover*
- *Need to take care which facts we give them*
- *Idea: order facts by relevance, give top n to prover ($n = 250, 1000, \ldots$)*
- *Meng & Paulson method: lightweight, symbol-based filter*
- *Machine learning method: look at previous proofs to get a probability of relevance*



From HOL to FOL

Source: higher-order, polymorphism, type classes
Target: first-order, untyped or simply-typed

- ***First-order:***
  - *SK combinators, λ-lifting*
  - *Explicit function application operator*
- ***Encode types:***
  - *Monomorphise (generate multiple instances), or*
  - *Encode polymorphism on term level*

Reconstruction

We don't want to trust the external provers.
Need to check/reconstruct proof.

- *Re-find using Metis*
  Usually fast and reliable (sometimes too slow)
- *Rerun external prover for trusted replay*
  Used for SMT. Re-runs prover each time!
- *Recheck stored explicit external representation of proof*
  Used for SMT, no need to re-run. Fragile.
- *Recast into structured Isar proof*
  Fast, not always readable.

Judgement Day (up to 2013)

Evaluating Sledgehammer:

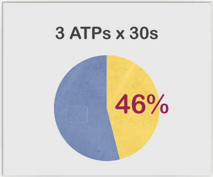
- *1240 goals out of 7 existing theories.*
- *How many can sledgehammer solve?*

- ***2010:*** *E, SPASS, Vampire (for 5-120s). 46%*
  $ESV \times 5s \approx V \times 120s$
- ***2011:*** *Add E-SInE, CVC2, Yices, Z3 (30s).*
  $Z3 > V$
- ***2012:*** *Better integration with SPASS. 64%*
  SPASS best (small margin)
- ***2013:*** *Machine learning for fact selection. 69%*
  Improves a few percent across provers.

Australian National University

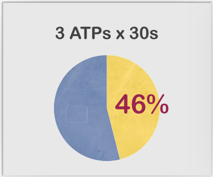
Evaluation

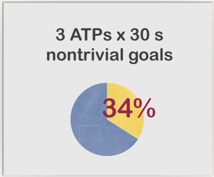
2010



Evaluation

2010





Australian National University

Evaluation

Judgement Day (2016)

Prover	MePo	MaSh	MeSh	Any selector
CVC4 1.5pre	679	749	**783**	830
E 1.8	622	601	**665**	726
SPASS 3.8ds	678	684	**739**	789
Vampire 3.0	703	698	**740**	789
veriT 2014post	543	556	**590**	655
Z3 4.3.2pre	638	668	**703**	788
Any prover	801	885	**919**	943

Fig. 15 Number of successful Sledgehammer invocations per prover on 1230 Judgment Day goals

919/1230 = **74%**

Sledgehammer rules!

Example application:

- *Large Isabelle/HOL repository of algebras for modelling imperative programs (Kleene Algebra, Hoare logic, ..., ≈ 1000 lemmas)*
- *Intricate refinement and termination theorems*
- *Sledgehammer and Z3 automate algebraic proofs at textbook level.*

"The integration of ATP, SMT, and Nitpick is for our purposes very very helpful."

Disproof

Theorem proving and testing

Testing can show only the presence of errors,
but not their absence. *(Dijkstra)*

Testing cannot prove theorems, but it can refute conjectures!

Sad facts of life:

- *Most lemma statements are wrong the first time.*
- *Theorem proving is expensive as a debugging technique.*

Find counter examples automatically!

Quickcheck

Lightweight validation by testing.

- *Motivated by Haskell's QuickCheck*
- *Uses Isabelle's code generator*
- *Fast*
- *Runs in background, proves you wrong as you type.*

Quickcheck

Covers a number of testing approaches:

- Random and exhausting testing.
- Smart test data generators.
- Narrowing-based (symbolic) testing.

Creates test data generators automatically.

Demo: Quickcheck

Test generators for datatypes

Fast iteration in continuation-passing-style

datatype α list = Nil | Cons α (α list)

Test function:

$\text{test}_{\alpha\ \mathit{list}}$ P = P Nil *andalso* test_{α} (λx. $\text{test}_{\alpha\ \mathit{list}}$ (λxs. P (Cons x xs)))

Test generators for predicates

$$\text{distinct xs} \Longrightarrow \text{distinct (remove1 x xs)}$$

Problem:
Exhaustive testing creates many useless test cases.

Solution:
Use definitions in precondition for smarter generator.
Only generate cases where distinct xs is true.

$\mathit{test\text{-}distinct}_{\alpha\ \mathit{list}}\ P\ =\ P\ \mathit{Nil\ andalso}$
$\mathit{test}_{\alpha}\ (\lambda x.\ \ \mathit{test\text{-}distinct}_{\alpha\ \mathit{list}}\ (\mathit{if}\ x \notin \mathit{xs}\ \mathit{then}\ (\lambda \mathit{xs}.\ \ P\ (\mathit{Cons}\ x\ \mathit{xs}))$
$\mathit{else\ True}))$

Use data flow analysis to figure out which variables
must be computed and which generated.

Narrowing

Symbolic execution with demand-driven refinement

- *Test cases can contain variables*
- *If execution cannot proceed: instantiate with further symbolic terms*

Pays off if large search spaces can be discarded:

distinct (Cons 1 (Cons 1 x))

False for any x, no further instantiations for x necessary.

Implementation:
Lazy execution with outer refinement loop.
Many re-computations, but fast.

Quickcheck Limitations

Only executable specifications!

- *No equality on functions with infinite domain*
- *No axiomatic specifications*

Nitpick

Nitpick

Finite model finder

- *Based on SAT via Kodkod (backend of Alloy prover)*
- *Soundly approximates infinite types*

Nitpick Successes

- *Algebraic methods*
- *C++ memory model*
- *Found soundness bugs in TPS and LEO-II*

Fan mail:

> *"Last night I got stuck on a goal I was sure was a theorem. After 5–10 minutes I gave Nitpick a try, and within a few secs it had found a splendid counterexample—despite the mess of locales and type classes in the context!"*

Demo: Nitpick

Automation Summary

- Proof: Sledgehammer
- Counter examples: Quickcheck
- Counter examples: Nitpick