**COMP 4161**
NICTA Advanced Course

**Advanced Topics in Software Verification**

Toby Murray, June Andronick, Gerwin Klein

**Slide 1**

## Content



[a] a1 due; [b] a2 due; [c] a3 due

**Slide 2**



## Overview

**Automatic Proof and Disproof**

- ➜ Sledgehammer: automatic proofs
- ➜ Quickcheck: counter example by testing
- ➜ Nipick: counter example by SAT

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

**Slide 3**

## 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, Yices, Z3

**The key:**

Efficient reasoning engines, and restricted logics.

**Slide 4**

## 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)

**Slide 5**

## Sledgehammer

**Sledgehammer:**

- ➜ Connects Isabelle with ATPs and SMT solvers:
  E, SPASS, Vampire, CVC3, 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

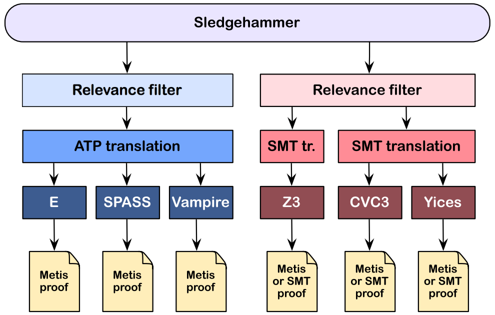
**Slide 6**



## DEMO: SLEDGEHAMMER

**Slide 7**

## Sledgehammer Architecture




**Slide 8**





NICTA 2013, provided under Creative Commons Attribution License, based on slides by J. Blanchette, L. Bulwahn and T. Nipkow

## Fact Selection

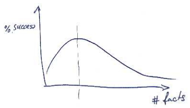
NICTA

**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



**Slide 9**

## From HOL to FOL

NICTA

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

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

**Slide 10**





## Reconstruction

NICTA

**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, experimental.

**Slide 11**

## Judgement Day

NICTA

**Evaluating Sledgehammer:**

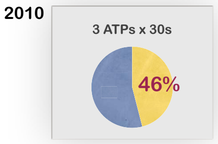
- ➜ 1240 goals out of 7 existing theories.
- ➜ How many can sledgehammer solve?

- ➜ **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.

**Slide 12**

## Evaluation

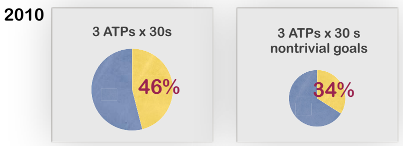
NICTA




**Slide 13**

## Evaluation

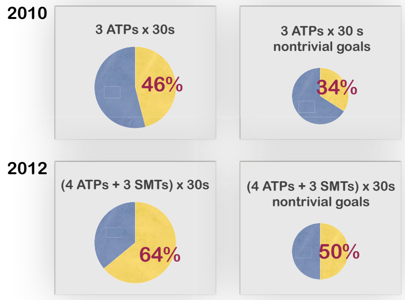
NICTA




**Slide 14**

## Evaluation

NICTA




**Slide 15**

## Sledgehammer rules!

NICTA

**Example application:**

- ➜ Large Isabelle/HOL repository of algebras for modelling imperative programs (Kleene Algebra, Hoare logic, . . ., $\approx 1000$ 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." – G. Struth

**Slide 16**

## DISPROOF

Slide 17

## 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!**

Slide 18





## 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.

Slide 19

## Quickcheck

**Covers a number of testing approaches:**

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

Creates test data generators automatically.

Slide 20

NICTA 2013, provided under Creative Commons Attribution License, based on slides by J. Blanchette, L. Bulwahn and T. Nipkow

## DEMO: QUICKCHECK

Slide 21

## Test generators for datatypes

**Fast iteration in continuation-passing-style**

**datatype** $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)

**Test function:**

test$_{\alpha\ list}$ P = P Nil *andalso* test$_{\alpha}$ ($\lambda$x. test$_{\alpha\ list}$ ($\lambda$xs. P (Cons x xs)))

Slide 22





## Test generators for predicates

distinct xs $\Longrightarrow$ 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.

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

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

Slide 23

## 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.

Slide 24

## Quickcheck Limitations

**Only executable specifications!**

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

**Slide 25**

# NITPICK

**Slide 26**

## Nitpick

**Finite model finder**

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

**Slide 27**

## 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!"

**Slide 28**

NICTA

## DEMO: NITPICK

Slide 29

NICTA

## We have seen today ...

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

Slide 30