Isabelle/HOL
Interaction and Automation

Jasmin Christian Blanchette
亚斯麦 - 克里斯蒂安 - 布兰切特
Technische Universität München

Early automated deduction (= theorem proving)

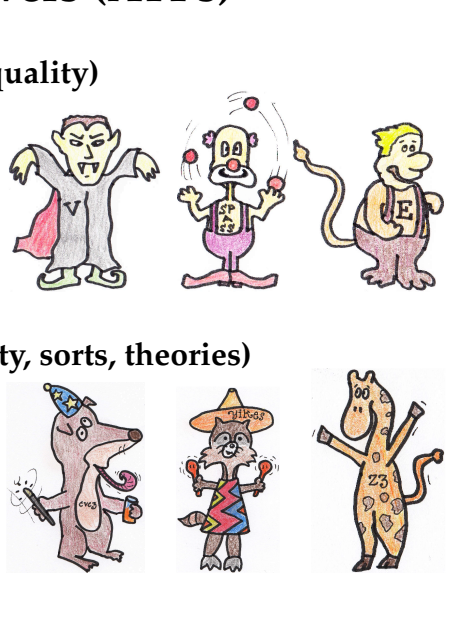
SAT solving (Davis, Putnam, Logeman, Loveland)
Resolution (Robinson)
LCF and ML (Milner)
NQTHM (Boyer & Moore)

Interactive theorem provers (= proof assistants)

ACL2 (Kaufmann & Moore)
Coq (Huet & Coquant)
HOL system (Gordon)
Isabelle (Paulson & Nipkow)
PVS (Owre & Shankar)

Automatic theorem provers (ATPs)

Traditional ATPs (for FOL with equality)
Otter (McCune)
Vampire (Voronkov)
SPASS (Weidenbach)
E (Schulz)



SMT solvers (for FOL with equality, sorts, theories)
CVC (Barrett & Tinelli)
Yices (Dutertre & de Moura)
Z3 (Bjørner & de Moura)

Higher-order ATPs
TPS (Andrews)
LEO-II (Benzmüller)
Isabelle
Satallax (Brown)
agsyHOL (Lindblad)

Isabelle—"The next 700 proof assistants"

Generic: Isabelle/{CTT, FOL, HOL, TLA, ZF, ...}

Two levels:
  metalogic shared by all instances
  object logic (CTT, FOL, HOL, ...)

HOL (higher-order logic) is the most developed object logic

Similar to languages like Standard ML, OCaml, Haskell
but total—cannot define $f\ n = f\ n + 1$
and can compare functions (even with infinite domains)

HOL hits a sweet spot:
  expressive enough for most maths and C.S.
  simple and easy-to-understand syntax and semantics
  convenient type system
  automatable

Demonstration

Why use proof assistants?

More reliable than a paper proof
  "Paper proof = pseudocode"
Extremely detailed (foundational)
Convenient for highly technical proofs
Convenient to experiment with variations
Fun and addictive

Why *not* use proof assistants?

Paper is usually much quicker
Formalizing well-understood theories yields few insights

Success stories

Proof assistants in general
Gödel's first incompleteness theorem (Shankar 1984)
Floating-point unit verification (AMD, Intel 1990s–now)
Four-color theorem (Gonthier 2008)
Verified compiler (Leroy 2009)
Kepler conjecture (Hales 201x)

Isabelle/HOL
Protocol verification (Paulson 1998)
Jinja (Klein & Nipkow 2005)
JinjaThreads (Lochbihler 2010)
seL4 microkernel (Klein 2009)
LTL model checker (Esparza et al. 2013)

Syntax of HOL

The *types* and *terms* of Isabelle are that of the simply typed λ-calculus augmented with constants, n-ary type constructors, and polymorphism

Types:		Terms:	
$\tau ::= {}'a$	type variable	$t ::= x^\tau$	variable
$\mid (\bar{\tau})\,\kappa$	type constructor	$\mid c^\tau$	constant
		$\mid \lambda x^\tau.\ t$	λ-abstraction
		$\mid t\ t'$	application

It is enough to consider *bool*, $'a \to {}'b$ (functions), and $=^{'a \to 'a \to bool}$ as built-in

Quantifiers and connectives can be defined:
  $\mathsf{True} \equiv (\lambda x.\, x) = (\lambda x.\, x)$
  $(\forall x.\, Px) \equiv \mathsf{All}\ P \equiv P = (\lambda x.\, \mathsf{True})$
  ⋮

Typing rules:

$$\frac{}{\vdash x^\tau : \tau} \qquad \frac{}{\vdash c^\tau : \tau} \qquad \frac{\vdash t : \upsilon}{\vdash \lambda x^\tau.\, t : \tau \to \upsilon} \qquad \frac{\vdash t : \tau \to \upsilon \quad \vdash u : \tau}{\vdash t\, u : \upsilon}$$

Theorems and proofs

Isabelle is implemented in Standard ML
(a safe typed functional language developed by Milner for LCF)

Theorems are represented by an opaque `thm` type
whose values are derived by inference primitives

All inferences ultimately happen in the kernel

Forward proof

Theorems can be derived by applying inferences to theorems

Backward proof

To prove a formula p:
1. start with theorem $p \Longrightarrow p$ (where the first p is the goal)
2. massage an assumption, yielding $p'_1 \Longrightarrow \cdots \Longrightarrow p'_n \Longrightarrow p$
3. repeat step 2 until $n = 0$

This is normally done using *tactics*,
which are joined together by *tacticals* (higher-order tactics)

ML examples

Forward proof

```
th = @{thm hd.simps};      (* hd (x # xs) = x *)
th' = th RS @{thm sym};    (* t = hd (t # xs) *)
```

Backward proof

```
goal = @{prop "p & q ==> q & p"};
tac = REPEAT (rtac @{thm conjI} 1 ORELSE
              etac @{thm conjE} 1 ORELSE
              atac 1);
th = Goal.prove @{context} [] [] goal (K tac);
```

Isar proofs

Structured version:

  lemma "$p \wedge q \Longrightarrow q \wedge p$"
  proof –
    assume pq: "$p \wedge q$"
    from pq **have** p: "p" **by** (rule conjunct1)
    from pq **have** q: "q" **by** (rule conjunct2)
    from q **and** p **show** "$q \wedge p$" **by** (rule conjI)
  qed

Script version:

  lemma "$p \wedge q \Longrightarrow q \wedge p$"
  apply (erule conjE)   (∗ 1. $p \Longrightarrow q \Longrightarrow q \wedge p$ ∗)
  apply (rule conjI)   (∗ 1. $p \Longrightarrow q \Longrightarrow q$ 2. $p \Longrightarrow q \Longrightarrow p$ ∗)
   apply assumption   (∗ 1. $p \Longrightarrow q \Longrightarrow p$ ∗)
  apply assumption   (∗ no subgoals! ∗)
  done

Mixed ML/script version:

  lemma "$p \wedge q \Longrightarrow q \wedge p$"
  by (tactic {∗
```
    REPEAT (rtac @{thm conjI} 1 ORELSE
            etac @{thm conjE} 1 ORELSE
            atac 1)
```
  ∗})

Compact version:

  lemma "$p \wedge q \Longrightarrow q \wedge p$"
  by auto

Automation

Simplifier
Classical reasoner
Combination (*auto*)
Tableau (*blast*)
Resolution prover (*metis*)
Linear arithmetic (*arith*)

Specification mechanisms

Low-level
Type declaration
Typedef
Axioms
Definitions

High-level
(Co)datatypes
(Co)inductive predicates
Terminating/tail-recursive recursive functions
Primitive corecursive functions

Isar examples

  typedecl *name*

  typedef $'a\ ne\_set$ = "$\{A :: {}'a\ set.\ A \neq \{\}\}$"
  by *auto*

  axiomatization name0 :: *name* **and** name1 :: *name* **where**
   name0_ne_1: "name0 $\neq$ name1"

  definition is_name0_or_1 :: "*name* $\Rightarrow$ *bool*" **where**
   "is_name0_or_1 $n \longleftrightarrow n$ = name0 $\vee$ n = name1"

  datatype $'a\ list$ = Nil | Cons $'a$ "$'a\ list$"

  codatatype $'a\ llist$ = LNil | LCons $'a$ "$'a\ llist$"

  fun len :: "$'a\ list \Rightarrow nat$" **where**
   "len Nil = 0" |
   "len (Cons _ xs) = Suc (len xs)"

  primcorec iterate :: "$('a \Rightarrow {}'a) \Rightarrow {}'a \Rightarrow {}'a\ llist$" **where**
   "iterate $f\ a$ = LCons a (iterate f ($f\ a$))"

More automation via external ATPs

1990s–now: Attempted many times
  either worked well for obscure systems
  or worked not so well for popular systems

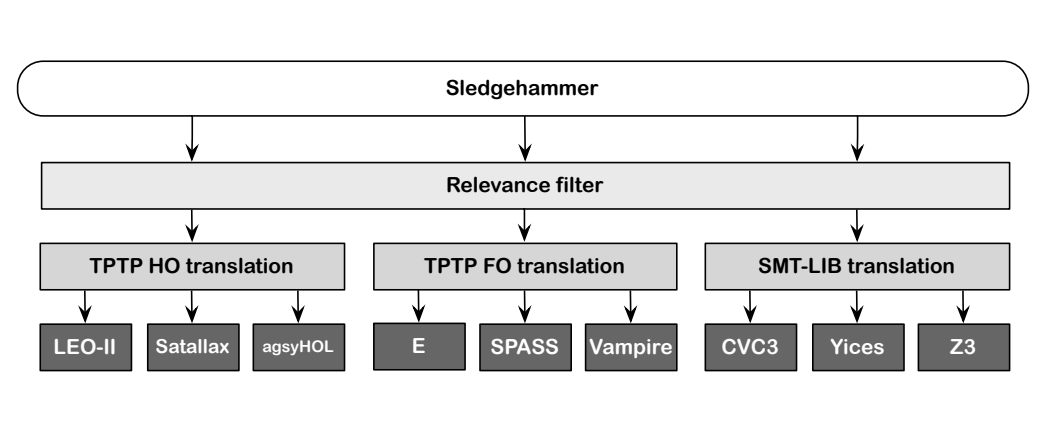
"Not so well"?
  limited to fragments
  heavy or unsound translations
  weak engineering
  poor ATP tuning
  no reconstruction

Sledgehammer (Paulson 2003–2009, B. 2010–now) is perhaps
the only tool of its kind that is used daily

Geoff Sutcliffe (Oct. 2013) wrote:

> *The following is a summary of the logged data for the three months July to September 2013. In that period there were 182 450 requests to SystemOnTPTP. Of those, 174 078 were to run ATP systems, the most popular being SInE 0.4 (**79 836 requests**) and Vampire 2.6 (**31 429 requests**). It is believed that **most of these requests come from users of Isabelle/HOL and Sledgehammer**.*

Sledgehammer's architecture



Relevance filters
MePo: Iterative, symbol-based
MaSh: Machine learning

ATP invocation
Locally or remotely (SystemOnTPTP)
Time slicing

Translation
(Incomplete) encoding of HO constructs in FOL
Encoding of types:
  monomorphization
  lightweight sound encodings

Reconstruction
One-liner *metis* call
Detailed Isar proof

Demonstration

Counterexample Generation

Quickcheck (2004–2012)
Inspired by Haskell QuickCheck
Idea: For fragment of HOL that fits in ML,
generate ML code and evaluate it for random values
(or bounded-exhaustively or by symbolic evaluation)

Nitpick (2009–now)
Based on Alloy's backend Kodkod, which itself is based on SAT
Idea: Generate first-order constraints and solve them using Kodkod

Success stories—Sledgehammer and Nitpick

JinjaThreads, seL4, LTL model checker
Algebraic methods (Gutmann et al. 2011)
C++ concurrency (B. et al. 2011)
Formal methods course (Genet 2012)
Modal logic & God's existence (Benzmüller 2010–now)

Some quotes

> Counterexample generators such as Nitpick complement the ATP systems and **allow a proof and refutation game** which is useful for developing and debugging formal specifications.
>   — Guttmann et al. (2011)

> Counterexamples found by Nitpick and Quickcheck permitted to debug rather complex functions. They were **intensively used by students**.
>   — Thomas Genet (2012)

> 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!
> Unfortunately I now have to revise my formalization `:-(`
>   — Tobias Nipkow (2011)

> I have recently been working on a new development. Sledgehammer has found some simply incredible proofs. I would estimate the **improvement in productivity** as a factor of **at least three, maybe five**.
>   — Lawrence C. Paulson (2012)

> S/h found an unsound proof and that way alerted me that two of my class axioms were inconsistent. **Cool!**
>   — Tobias Nipkow (2013)

> Since I started using Isabelle (Isabelle 2009-1?) sledgehammer has gone from being an unexpected surprise when it came up with an answer, to being a **significant tool in my daily workflow**.
>   — David Greenaway (2013)

```
(*This homework was presented to you by:
   _____ _         _           _
  |   __| |___ _| |___ ___| |_ ___ _____ _____ ___ ___
  |__   | | -_| . | . | -_|   | .'|     |     | -_|  _|
  |_____|_|___|___|_  |___|_|_|__,|_|_|_|_|_|_|___|_|
                  |___|
*)
```
  — A student (2013)

Summary

  powerful external tools $\wedge$
  sound, efficient translations $\wedge$
  one-click invocation $\Longrightarrow$
   productive users