Extending Sledgehammer with Induction Provers

Jasmin Blanchette

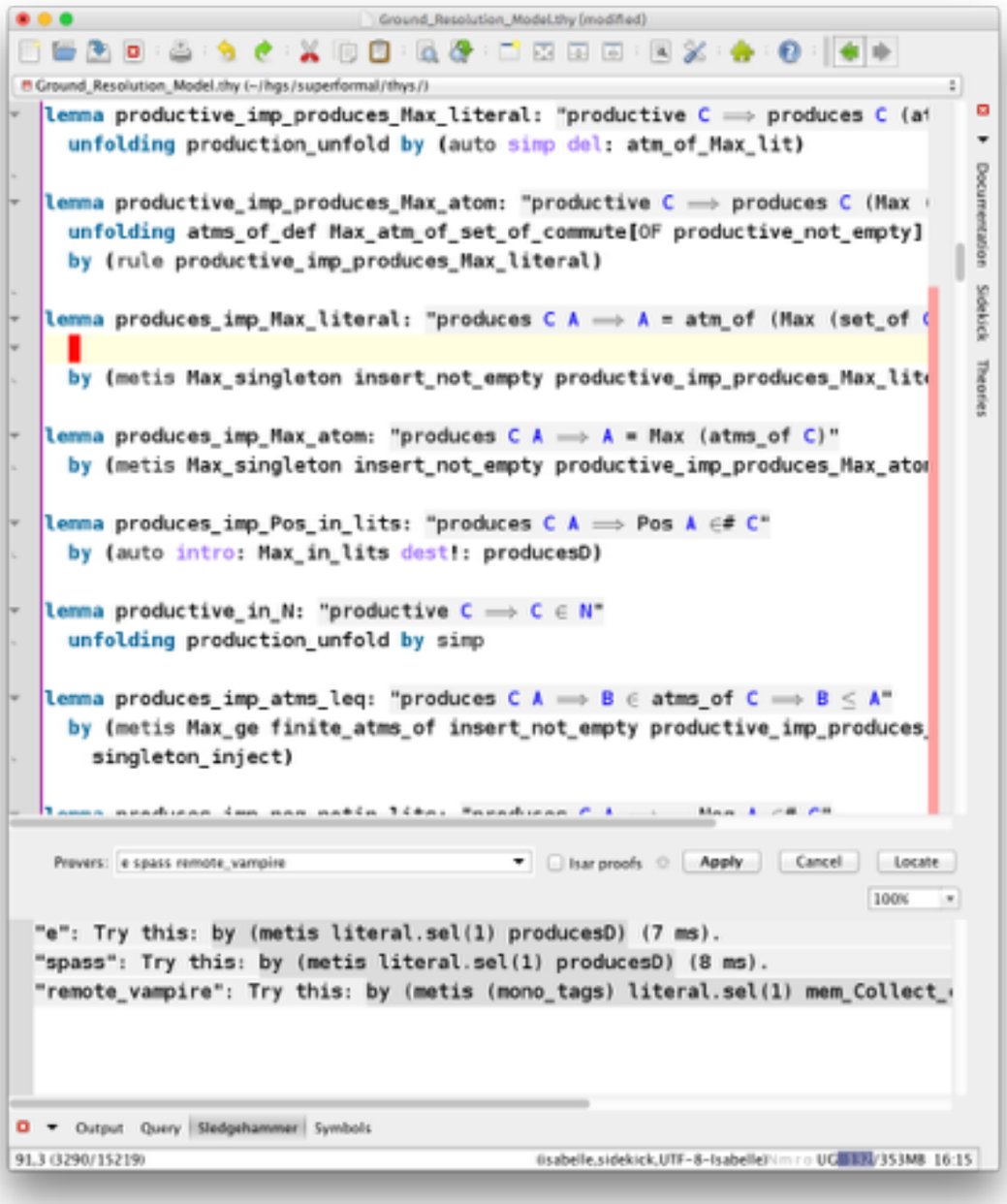
Inria & LORIA, Nancy
MPI für Informatik, Saarbrücken

Isabelle

- **general purpose** proof assistant
- HOL: **higher-order logic**
- **small** kernel
- **safe** definitional principles
- **high** proof automation
- **modern** user interface

Isabelle

- general purpose proof assistant
- HOL: higher-order logic
- small kernel
- safe definitional principles
- high proof automation
- modern user interface



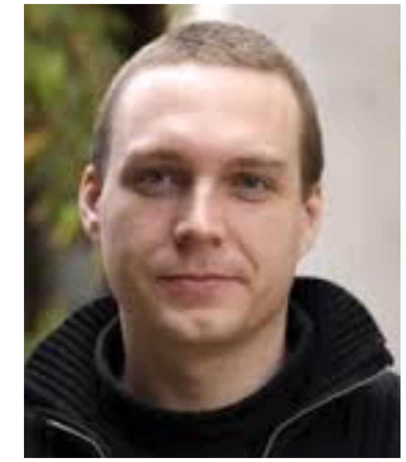
Automatic Proofs and Counterexamples

- **Sledgehammer** finds proofs using external ATPs
  CVC4, E, LEO-II, Satallax, SPASS, Vampire, veriT, Z3, …
- **Nitpick** finds counterexamples using external model finder
  Kodkod (Alloy's backend)
- **Quickcheck** finds counterexamples using random testing, exhaustive testing, narrowing

powerful automation $\Longrightarrow$ productive users

Larry Paulson

Sledgehammer has found some incredible proofs. I would estimate the improvement in productivity as a factor of **at least three, maybe five**.



Gerwin Klein

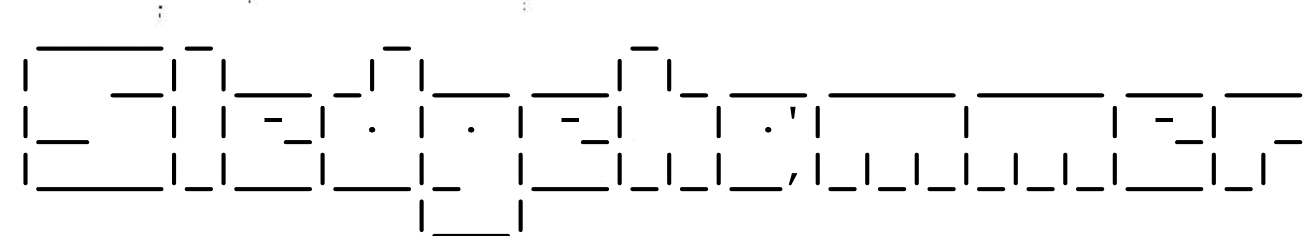
Nitpick (sehr nützlich) hat mich nach ungefähr der Hälfte überzeugt, dass meine Idee mit dem Merge nicht funktioniert.

Tobias Nipkow

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**! Unfortunately I now have to revise my formalization :-(

Developing proofs without Sledgehammer is like **walking** as opposed to **running**.

```
(*This homework was presented to you by:
```



```
*)
```

Sledgehammer-Generated Proofs

have *f1*: $\forall n.$ rat_of_int $\lfloor$rat_of_nat $n\rfloor =$ rat_of_nat n
  using *of_int_of_nat_eq* **by** *simp*
have *f2*: $\forall n.$ real_of_int $\lfloor$rat_of_nat $n\rfloor =$ real n
  using *of_int_of_nat_eq real_eq_of_nat* **by** *auto*
have *f3*: $\forall i\ ia.\ \lfloor$rat_of_int i / rat_of_int $ia\rfloor = \lfloor$real_of_int i / real_of_int $ia\rfloor$
  using *div_is_floor_divide_rat div_is_floor_divide_real* **by** *simp*
have *f4*: $0 < \lfloor$rat_of_nat $p\rfloor$
  using p **by** *simp*
have $\lfloor S \rfloor \leq s$
  using *less floor_le_iff* **by** *auto*
hence $\lfloor$rat_of_int $\lfloor S \rfloor$ / rat_of_nat $p\rfloor \leq \lfloor$rat_of_int s / rat_of_nat $p\rfloor$
  using *f1 f3 f4* **by** (*metis div_is_floor_divide_real zdiv_mono1*)
hence $\lfloor S$ / real $p\rfloor \leq \lfloor$rat_of_int s / rat_of_nat $p\rfloor$
  using *f1 f2 f3 f4* **by** (*metis div_is_floor_divide_real floor_div_pos_int*)
thus S / real $p \leq$ real_of_int $(s$ div int $p) + 1$
  using *f1 f3*
  by (*metis div_is_floor_divide_real floor_le_iff floor_of_nat less_eq_real_def*)

Sledgehammer-Generated Proofs

lemma *powsum_ub:* $i \leq n \Longrightarrow x^i \leq x_0^n$
proof (*induct n*)
  case 0 **show** *case*
    by (*metis* (*hide_lams*, *mono_tags*) *0.prems eq_iff le_0_eq power_0 powsum_00*)
next
  case (Suc *n*) **show** *case*
proof −
  { **assume** *aa1*: Suc $n \neq i$
    have *ff1*: $x_0^{\mathsf{Suc}\,n} \leq x_0^{\mathsf{Suc}\,n} \wedge \mathsf{Suc}\,n \neq i$
      using *aa1* **by** *fastforce*
    have *ff2*: $\exists a.\ x_0^n + a \leq x_0^{\mathsf{Suc}\,n} \wedge \mathsf{Suc}\,n \neq i$
      using *ff1 powsum2* **by** *auto*
    have $x^i \leq x_0^{\mathsf{Suc}\,n}$
      using *ff2* **by** (*metis Suc.hyps Suc.prems add_lub le_SucE less_eq_def*) }
  thus $x^i \leq x_0^{\mathsf{Suc}\,n}$
    using *less_eq_def powsum_split_var2* **by** *auto*
qed
qed

Wish List for Prover Authors

- **detailed output** similar to input syntax
- record which induction **principle** and **instance** were used
- respect the **Russian doll** principle
- well-documented input
- rich input

Datatypes

- Polymorphism:

  datatype 'a list = Nil | Cons 'a ('a list)

- Mutual recursion:

  datatype 'a tree = Node 'a ('a trees)
  and 'a trees = TNil | TCons ('a tree) ('a trees)

- Nested recursion:

  datatype 'a tree = Node 'a ('a tree list)

  fset?

  llist?

Codatatypes

- Polymorphism:

  codatatype 'a llist = LNil | LCons 'a ('a llist)

- Mutual recursion:

  codatatype 'a ltree = LNode 'a ('a ltrees)
  and 'a ltrees = LTNil | LTCons ('a ltree) ('a ltrees)

- Nested recursion:

  codatatype 'a ltree = LNode 'a ('a ltree llist)
  fset?
  list?

Non-Free Datatypes?

Rule (Co)induction?

Intergration in a HO ATP? Work with Arbitrary Induction Principles?

Syntax for Benchmarks

- Should be **natural**, not an encoding
- Ideally, extension of something existing
  - TIS syntax!
  - Theories?
  - Important to have a parser/AST
  - Checker

Thank you!