Jasmin Christian Blanchette

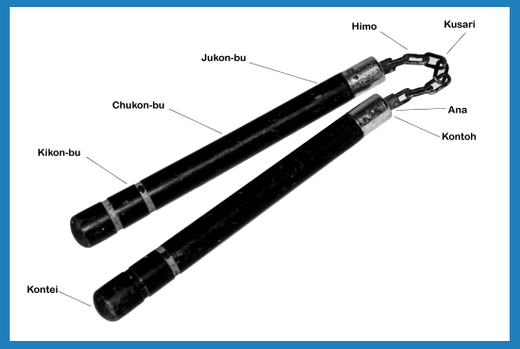
So what are hammers (and counterexample generators) good for?

VU VRIJE UNIVERSITEIT AMSTERDAM

Talk outline

1. Sledgehammer

2. Nitpick



3. Nunchaku

4. Lean Forward

1. **Sledgehammer**
Automatic proof search
for Isabelle/HOL

Joint work with
Sascha Böhme, Jia Meng, Tobias Nipkow,
Larry Paulson, Makarius Wenzel, and many others

Does there exist a function f from reals to reals such that for all x and y, $f(x + y^2) - f(x) \geq y$?

```
let lemma = prove
(`!f:real->real. ~(!x y. f(x + y * y) - f(x) >= y)`,
 REWRITE_TAC[real_ge] THEN REPEAT STRIP_TAC THEN
 SUBGOAL_THEN `!n x y. &n * y <= f(x + &n * y * y) - f(x)` MP_TAC THENL
  [MATCH_MP_TAC num_INDUCTION THEN SIMP_TAC[REAL_MUL_LZERO; REAL_ADD_RID] THEN
   REWRITE_TAC[REAL_SUB_REFL; REAL_LE_REFL; GSYM REAL_OF_NUM_SUC] THEN
   GEN_TAC THEN REPEAT(MATCH_MP_TAC MONO_FORALL THEN GEN_TAC) THEN
   FIRST_X_ASSUM(MP_TAC o SPECL [`x + &n * y * y`; `y:real`]) THEN
   SIMP_TAC[REAL_ADD_ASSOC; REAL_ADD_RDISTRIB; REAL_MUL_LID] THEN
   REAL_ARITH_TAC;
   X_CHOOSE_TAC `m:num` (SPEC `f(&1) - f(&0):real` REAL_ARCH_SIMPLE) THEN
   DISCH_THEN(MP_TAC o SPECL [`SUC m EXP 2`; `&0`; `inv(&(SUC m))`]) THEN
   REWRITE_TAC[REAL_ADD_LID; GSYM REAL_OF_NUM_SUC; GSYM REAL_OF_NUM_POW] THEN
   REWRITE_TAC[REAL_FIELD `(&m + &1) pow 2 * inv(&m + &1) = &m + &1`;
     REAL_FIELD `(&m + &1) pow 2 * inv(&m + &1) * inv(&m + &1) = &1`] THEN
   ASM_REAL_ARITH_TAC]);;
```

John Harrison

Does there exist a function f from reals to reals such that for all x and y, $f(x + y^2) - f(x) \geq y$?

[1] $f(x + y^2) - f(x) \geq y$ for any x and y (given)

[2] $f(x + ny^2) - f(x) \geq ny$ for any x, y, and natural number n
(by an easy induction using [1] for the step case)

[3] $f(1) - f(0) \geq m + 1$ for any natural number m
(set $n = (m + 1)^2$, $x = 0$, $y = 1/(m + 1)$ in [2])

[4] Contradiction of [3] and the Archimedean property of the reals

John Harrison

```
lemma
  shows "¬ (∃f :: real ⇒ real. ∀x y. f (x + y * y) - f x ≥ y)"
proof
  assume "∃f :: real ⇒ real. ∀x y. f (x + y * y) - f x ≥ y"
  then obtain f :: "real ⇒ real" where f: "∧x y. f (x + y * y) - f x ≥ y"
    by blast
  have nf: "∧(n :: nat) x y. f (x + real n * y * y) - f x ≥ real n * y"
  proof -
    fix n x y
    show "f (x + real n * y * y) - f x ≥ real n * y"
    proof (induct n)
      case 0 thus ?case by simp
    next
      case (Suc n) show ?case
      proof simp
        have "∃r. y ≤ f (y * y + (x + y * (y * real n))) - r ∧ y * real n ≤ r - f x"
        by (metis Suc.hyps add.commute f mult.commute)
      then have "y + y * real n ≤ f (y * y + (x + y * (y * real n))) - f x"
        by linarith
      then show "(1 + real n) * y ≤ f (x + (1 + real n) * y * y) - f x"
        by (simp add: add.left_commute distrib_left mult.commute)
      qed
    qed
  qed
  have min: "∧m. f 1 - f 0 ≥ real m + 1"
  proof -
    fix m
    show "f 1 - f 0 ≥ real m + 1"
    proof -
      have "∧r ra rb. (r :: real) / ra * rb = r * (rb / ra)"
        by simp
      then have "real (m + 1) * (real (m + 1) / real (m + 1)) ≤
          f (real (m + 1) * (real (m + 1) / (real (m + 1) * real (m + 1)))) - f 0"
        using nf[where n = "(m + 1) * (m + 1)" and x = 0 and y = "1 / (m + 1)"]
        by (metis (no_types) add.left_neutral divide_divide_eq_left mult.right_neutral of_nat_mult
          times_divide_eq_right)
      then have "real (m + 1) ≤ f 1 - f 0"
        by simp
      then show ?thesis
        by simp
    qed
  qed
  then show False
    by (metis add.commute add_le_imp_le_diff add_le_same_cancel2 add_mono diff_add_cancel
      ex_le_of_nat not_one_le_zero)
qed
```

intermediate properties

manual

generated automatically

Scratch.thy (modified)

Scratch.thy (~/)

```
lemma "A + B = (A #∪ B) + (A #∩ B)"
```

Proof state | Auto update | Update | Search: | 100%

```
proof (prove)
goal (1 subgoal):
 1. A + B = A #∪ B + A #∩ B
```

Output | Query | Sledgehammer | Symbols

Documentation | Sidekick | State | Theories

Scratch.thy (modified)

Scratch.thy (~/)

```
lemma "A + B = (A #∪ B) + (A #∩ B)"
sledgehammer
```

Proof state | Auto update | Update | Search: | 100%

```
Sledgehammering...
```

Output | Query | Sledgehammer | Symbols

Documentation | Sidekick | State | Theories

Scratch.thy (modified)

Scratch.thy (~/)

```
lemma "A + B = (A #∪ B) + (A #∩ B)"
sledgehammer
```

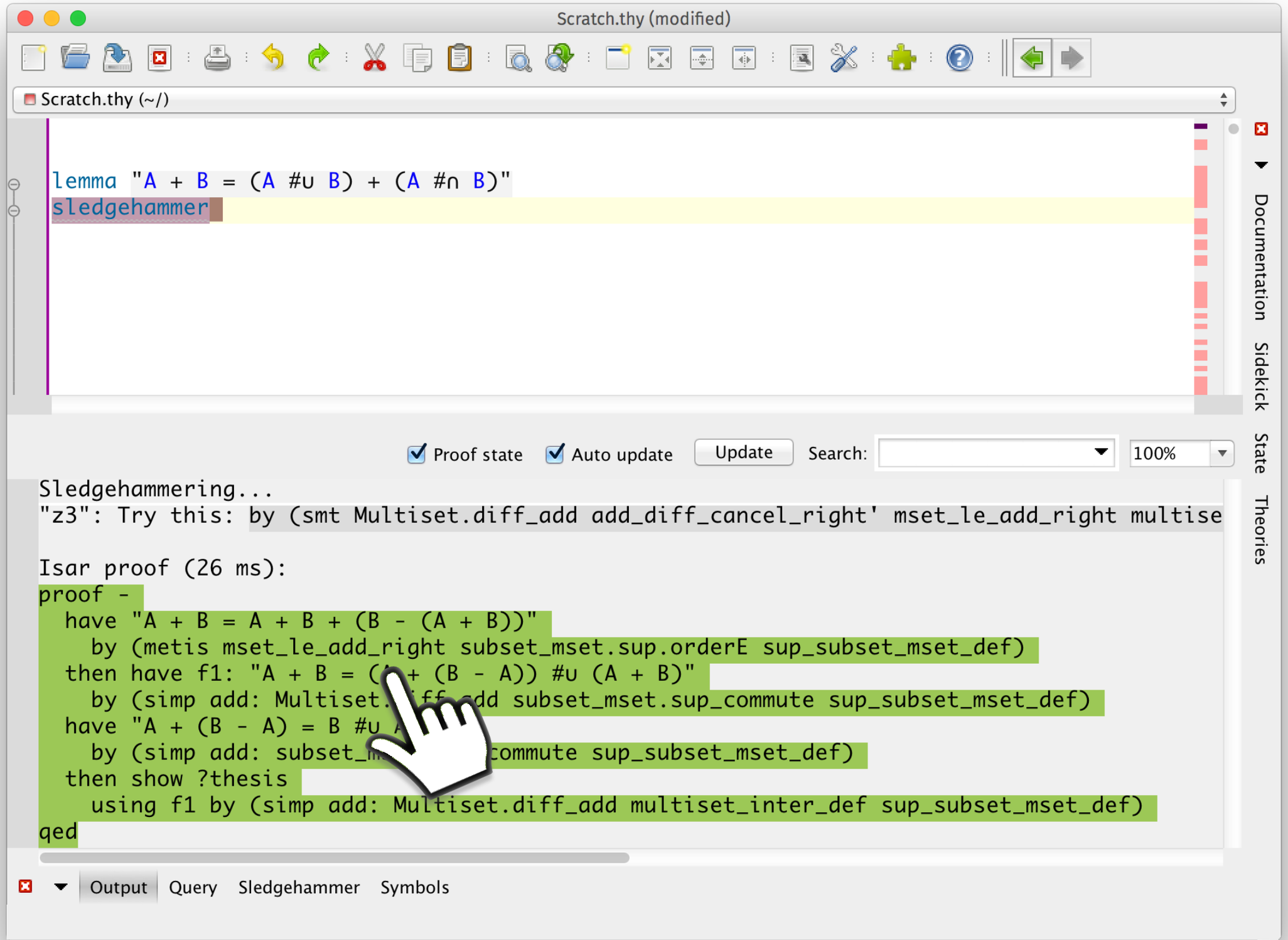
Proof state | Auto update | Update | Search: | 100%

```
Sledgehammering...
"z3": Try this: by (smt Multiset.diff_add add_diff_cancel_right' mset_le_add_right multise

Isar proof (26 ms):
proof -
  have "A + B = A + B + (B - (A + B))"
    by (metis mset_le_add_right subset_mset.sup.orderE sup_subset_mset_def)
  then have f1: "A + B = (A + (B - A)) #∪ (A + B)"
    by (simp add: Multiset.diff_add subset_mset.sup_commute sup_subset_mset_def)
  have "A + (B - A) = B #∪ A"
    by (simp add: subset_mset.sup_commute sup_subset_mset_def)
  then show ?thesis
    using f1 by (simp add: Multiset.diff_add multiset_inter_def sup_subset_mset_def)
qed
```

Output | Query | Sledgehammer | Symbols

Documentation | Sidekick | State | Theories


Scratch.thy (modified)
Scratch.thy (~/)
lemma "A + B = (A #∪ B) + (A #∩ B)"
sledgehammer
Proof state
Auto update
Update
Search:
100%
Sledgehammering...
"z3": Try this: by (smt Multiset.diff_add add_diff_cancel_right' mset_le_add_right multise
Isar proof (26 ms):
proof -
have "A + B = A + B + (B - (A + B))"
by (metis mset_le_add_right subset_mset.sup.orderE sup_subset_mset_def)
then have f1: "A + B = (A + (B - A)) #∪ (A + B)"
by (simp add: Multiset.diff_add subset_mset.sup_commute sup_subset_mset_def)
have "A + (B - A) = B #∪
by (simp add: subset_m commute sup_subset_mset_def)
then show ?thesis
using f1 by (simp add: Multiset.diff_add multiset_inter_def sup_subset_mset_def)
qed
Output
Query
Sledgehammer
Symbols
Documentation
Sidekick
State
Theories

Scratch.thy (modified)

Scratch.thy (~/)

```
lemma "A + B = (A #∪ B) + (A #∩ B)"
proof -
  have "A + B = A + B + (B - (A + B))"
    by (metis mset_le_add_right subset_mset.sup.orderE sup_subset_mset_def)
  then have f1: "A + B = (A + (B - A)) #∪ (A + B)"
    by (simp add: Multiset.diff_add subset_mset.sup_commute sup_subset_mset_def)
  have "A + (B - A) = B #∪ A"
    by (simp add: subset_mset.sup_commute sup_subset_mset_def)
  then show ?thesis
```

Proof state | Auto update | Update | Search: | 100%

```
proof (state)
goal (1 subgoal):
 1. A + B = A #∪ B + A #∩ B
```

Output | Query | Sledgehammer | Symbols

Documentation | Sidekick | State | Theories

Sledge_Ex.thy

Sledge_Ex.thy (~/hgs/inria/talks/coq2015-toward/thys/)

```
lemma "sum_n n = n * (n + 1) div 2"
proof (induction n)
  case 0
  show "sum_n 0 = 0 * (0 + 1) div 2"
  proof -
    have 1: "sum_n 0 = 0" by simp
    have 2: "(0::nat) * (0 + 1) div 2 = 0" by simp
    show ?thesis by (subst 1, subst 2) (rule refl)
  qed
next
  case (Suc n)
  assume IH: "sum_n n = n * (n + 1) div 2"
  show "sum_n (Suc n) = Suc n * (Suc n + 1) div 2"
  proof -
    have "sum_n (Suc n) = Suc n + sum_n n" using sum_n.simps(2) by metis
    also have "… = Suc n + n * (n + 1) div 2" apply (subst IH) by blast
    also have "… = (2 * Suc n + n * (n + 1)) div 2"
      using div_mult_self4 eq_numeral_simps(4) by metis
    also have "… = (Suc n * (2 + n)) div 2"
      using algebra_simps(3) Suc_eq_plus1 mult.commute by metis
    also have "… = (Suc n * (n + 2)) div 2" using add.commute by metis
    finally show ?thesis
      using Suc_eq_plus1 Suc_eq_plus1_left add.left_commute one_add_one
      by metis
```

Documentation Sidekick Theories

Output Query Sledgehammer Symbols

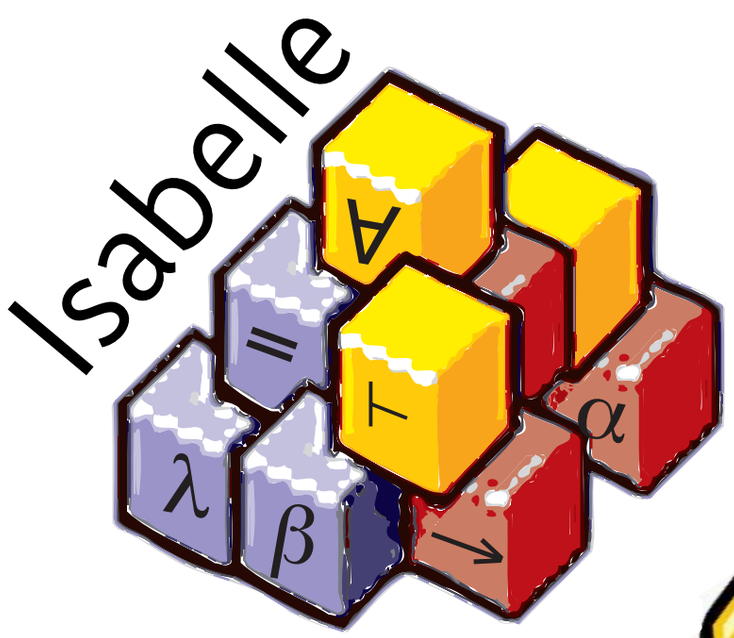
196,7 (5666/18833) (isabelle,sidekick,UTF-8-Isabelle)Nmro UG 243/547MB 16:05

Larry Paulson

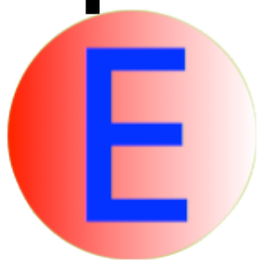
Sledgehammer has certainly transformed the way Isabelle is **taught**. There are two reasons for this:

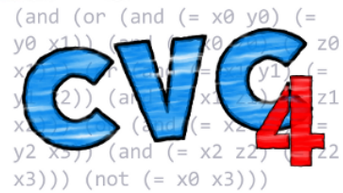
- Because it identifies relevant facts, users no longer need to **memorise lemma libraries**.
- Because it works in harmony with Isar structured proofs, users no longer need to **learn many low-level tactics**.

Proof assistants



Automatic provers





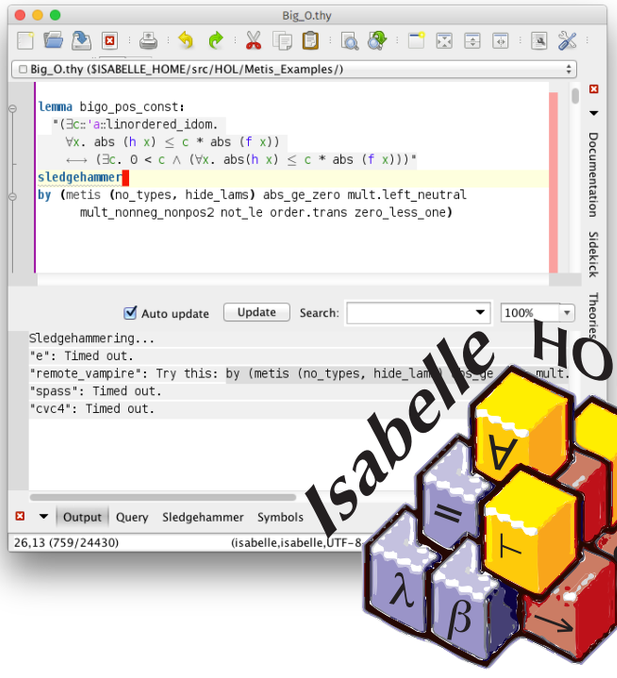
well suited for
large formalizations

but
require intensive
manual labor

Sledge-hammer

fully automatic

but
no proof
management



select lemmas +
translate to FOL

reconstruct proof

refutational

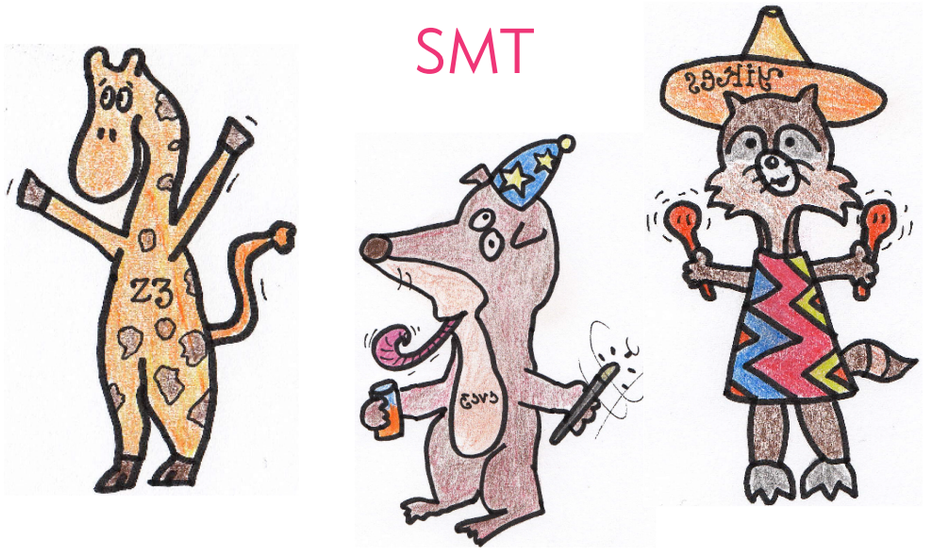
resolution rule

term ordering

equality reasoning

redundancy criterion

E, SPASS, Vampire, …



refutational

SAT solver

+ congruence closure

+ quantifier instantiation

+ other theories (e.g. LIA, LRA)

CVC4, veriT, Yices, Z3, …

Upon success,
proofs are translated to Isabelle

```
by (metis distinct_remdups finite_list set_remdups)
```

one-line

```
proof -
  assume a1: "finite A"
  have "∀A. ∃as. set (as::'a list) = A ∨ ¬ finite A"
    by (meson finite_list)
  then obtain aas :: "'a set ⇒ 'a list" where
    "∧A. ¬ finite A ∨ set (aas A) = A"
    by metis
  hence "set (aas A) = A"
    using a1 by auto
  hence "∃as. set as = A"
    by blast
  hence "∃as. distinct (remdups as) ∧ set as = A"
    by simp
  thus ?thesis
    by (meson set_remdups)
qed
```

detailed (Isar)

One-line proofs

```
lemma "length (tl xs) ≤ length xs"
by (metis diff_le_self length_tl)
```

proof method

lemmas

⊕ usually fast and reliable

⊕ lightweight

⊖ cryptic

⊖ sometimes slow (several seconds)

⊖ often cannot deal with theories

Detailed (Isar) proofs

```
lemma "length (tl xs) ≤ length xs"
proof -
  have "∧x1 x2. (x1::nat) - x2 - x1 = 0 - x2"
    by (metis comm_monoid_diff_class.diff_cancel diff_right_commute)
  hence "length xs - 1 - length xs = 0"
    by (metis zero_diff)
  hence "length xs - 1 ≤ length xs"
    by (metis diff_is_0_eq)
  thus "length (tl xs) ≤ length xs"
    by (metis length_tl)
qed
```

⊕ faster than one-liners

⊕ higher reconstruction success rate

⊕ self-explanatory?

⊖ technically more challenging

⊖ ugly?

```
lemma
  shows "¬ (∃f :: real ⇒ real. ∀x y. f (x + y * y) - f x ≥ y)"
proof
  assume "∃f :: real ⇒ real. ∀x y. f (x + y * y) - f x ≥ y"
  then obtain f :: "real ⇒ real" where f: "∧x y. f (x + y * y) - f x ≥ y"
    by blast
  have nf: "∧(n :: nat) x y. f (x + real n * y * y) - f x ≥ real n * y"
  proof -
    fix n x y
    show "f (x + real n * y * y) - f x ≥ real n * y"
    proof (induct n)
      case 0 thus ?case by simp
    next
      case (Suc n) show ?case
      proof simp
        have "∃r. y ≤ f (y * y + (x + y * (y * real n))) - r ∧ y * real n ≤ r - f x"
        by (metis Suc.hyps add.commute f mult.commute)
      then have "y + y * real n ≤ f (y * y + (x + y * (y * real n))) - f x"
        by linarith
      then show "(1 + real n) * y ≤ f (x + (1 + real n) * y * y) - f x"
        by (simp add: add.left_commute distrib_left mult.commute)
      qed
    qed
  qed
  have min: "∧m. f 1 - f 0 ≥ real m + 1"
  proof -
    fix m
    show "f 1 - f 0 ≥ real m + 1"
    proof -
      have "∧r ra rb. (r :: real) / ra * rb = r * (rb / ra)"
        by simp
      then have "real (m + 1) * (real (m + 1) / real (m + 1)) ≤
          f (real (m + 1) * (real (m + 1) / (real (m + 1) * real (m + 1)))) - f 0"
        using nf[where n = "(m + 1) * (m + 1)" and x = 0 and y = "1 / (m + 1)"]
        by (metis (no_types) add.left_neutral divide_divide_eq_left mult.right_neutral of_nat_mult
          times_divide_eq_right)
      then have "real (m + 1) ≤ f 1 - f 0"
        by simp
      then show ?thesis
        by simp
    qed
  qed
  then show False
    by (metis add.commute add_le_imp_le_diff add_le_same_cancel2 add_mono diff_add_cancel
      ex_le_of_nat not_one_le_zero)
qed
```

Sledgehammer really works

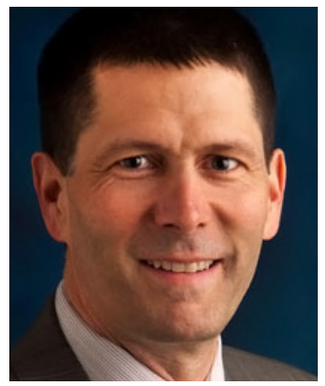
Developing proofs without Sledgehammer is like **walking as opposed to running**.

Tobias Nipkow

Larry Paulson

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**.

Sledgehammers … have led to visible success. Fully automated procedures can prove … 47% of the HOL Light/Flyspeck libraries, with comparable rates in Isabelle. These automation rates represent an **enormous saving in human labor**.



Thomas Hales

Isabelle's pros and cons, according to my students

⊕

11.5 Sledgehammer
4 Nitpick
4 Isar
2.5 automation
2 IDE
1 Quickcheck
1 set theory
1 schematic variables
1 structural induction
1 classical logic
1 function induction
1 infix operators
1 "qed auto"

⊖

5 goal/assumption handling
4 weak logic (props as types, types as terms)
3 Sledgehammer on lists, HO goals, or induction
1 automatic induction
1 Sledgehammer-generated Isar
1 arithmetic
1 Isar
1 opaque proofs
1 double quotes around inner syntax
1 underdeveloped "fset"
1 proof reuse
1 no hnf for statements, not even definitions
1 guaranteed computability
1 forward "apply" in assumptions (drule?)
1 error messages in inner syntax
1 ltac (Eisbach?)
1 cannot click on fun to see definition (?)
1 tooltips for built-in functions etc.

Sledgehammer's main weaknesses

⊖ Higher-order "lost in translation"

⊖ No induction

⊖ Explosive search space

matryoshka

2. **Nitpick**
A (counter)model finder
for Isabelle/HOL

Joint work with
Alexander Krauss and Tobias Nipkow

Nit_Ex.thy

Nit_Ex.thy

```
lemma exec_append: "exec (is1 @ is2) s stk = exec is2 s (exec is1 s stk)"
```

Documentation Sidekick Theories

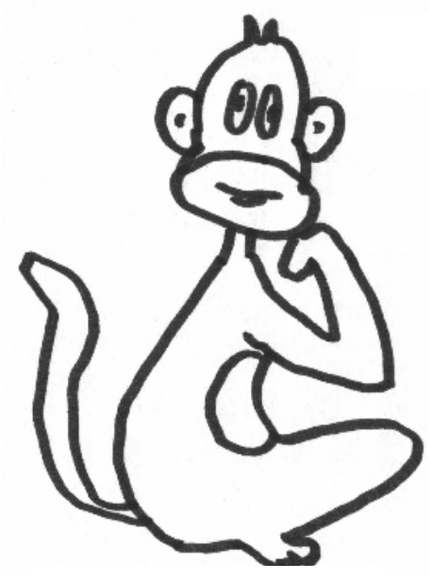
Auto update Update Search: 100%

Architecture

HOL

Isabelle

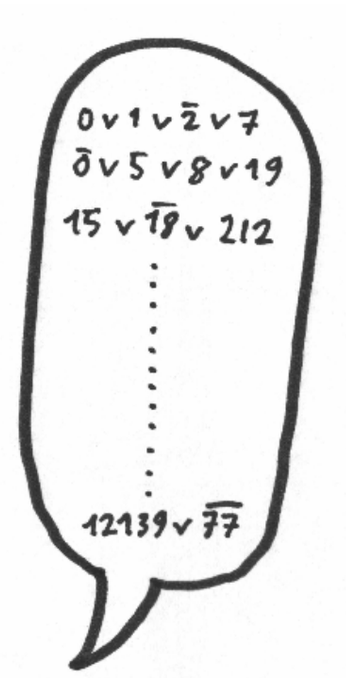
FORL



Nitpick

Kodkod

SAT



SAT solver

Translation

fixed finite? cardinalities:
try all cards. ≤ K for base types

first-order

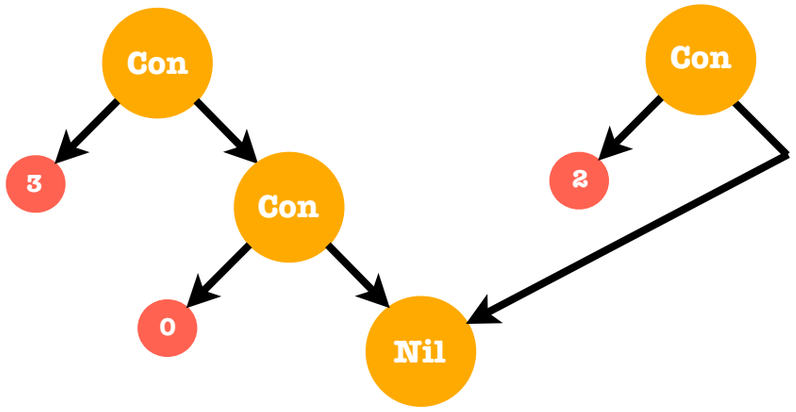
$\tau_1 \to \cdots \to \tau_n \to \text{bool} \longmapsto A_1 \times \cdots \times A_n$

$\tau_1 \to \cdots \to \tau_n \to \tau \longmapsto A_1 \times \cdots \times A_n \times A$
$+$ constraint

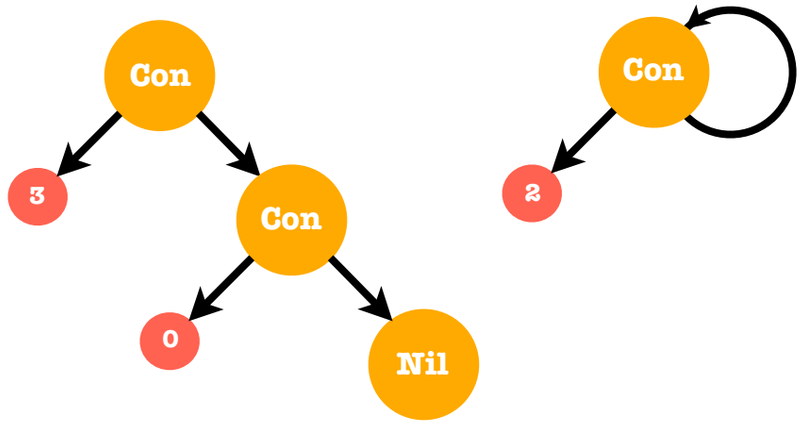
higher-order

$\sigma \to \tau \longmapsto \underbrace{A \times \cdots \times A}_{|\sigma| \text{ times}}$

Translation



datatypes



codatatypes

$p = F\ p$

$p_0 = (\lambda x.\ \mathrm{False})$

$p_{i+1} = F\ p_i$

inductive preds.

$p = F\ p$

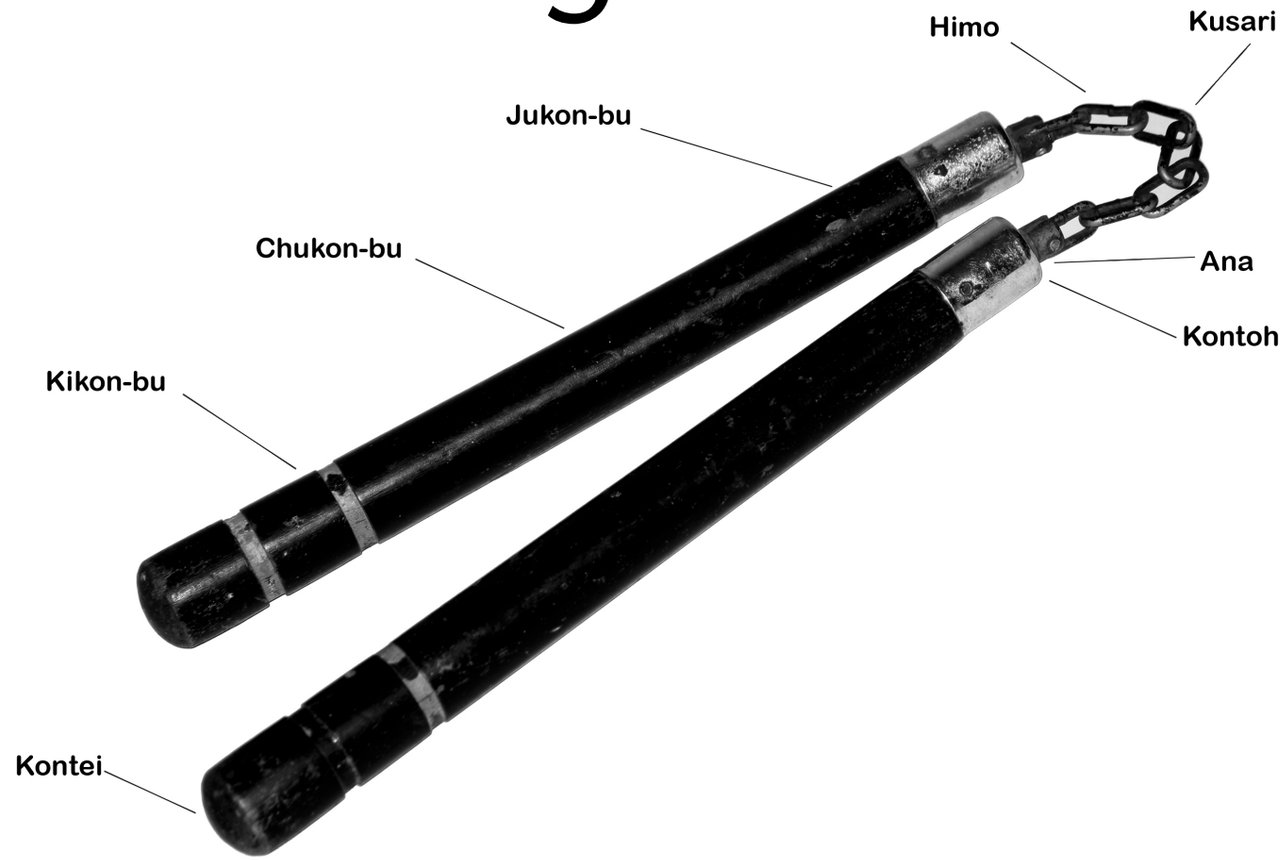
$p_0 = (\lambda x.\ \mathrm{True})$

$p_{i+1} = F\ p_i$

coinductive preds.

3. **Nunchaku**

A modular model finder for higher-order logic



Ongoing joint work with
Simon Cruanes, Pablo Le Hénaff, and Andrew Reynolds

multiple frontends

Isabelle/HOL, Lean, Coq, TLAPS, …

multiple backends

CVC4, Kodkod, Paradox, SMBC, Leon, Vampire, …

more precision

by better approximations

more efficiency

by using better backends and
by letting them enumerate cardinalities

Simplified translation pipeline

1. Monomorphize
2. Specialize
3. Polarize
4. Encode (co)inductive predicates
5. Encode (co)recursive functions
6. Encode higher-order functions

Actual translation pipeline

```
$ nunchaku --print-pipeline
Pipeline:
  ty_infer ➜ convert ➜ skolem ➜
  fork {
    mono ➜ elim_infinite ➜ elim_copy ➜ elim_multi_eqns ➜ specialize ➜ elim_match ➜ elim_codata ➜
    polarize ➜ unroll ➜ skolem ➜ elim_ind_pred ➜ elim_quant ➜ lift_undefined ➜ model_clean ➜
    close {smbc ➜ id}
  | mono ➜ elim_infinite ➜ elim_copy ➜ elim_multi_eqns ➜ specialize ➜ elim_match ➜
    fork {
      elim_codata ➜ polarize ➜ unroll ➜ skolem ➜ elim_ind_pred ➜ elim_data ➜ lambda_lift ➜ elim_hof ➜
      elim_rec ➜ intro_guards ➜ elim_prop_args ➜
      fork {
        elim_types ➜ model_clean ➜ close {to_fo ➜ elim_ite ➜ conv_tptp ➜ paradox ➜ id}
      | model_clean ➜ close {to_fo ➜ fo_to_rel ➜ kodkod ➜ id}
      }
    | polarize ➜ unroll ➜ skolem ➜ elim_ind_pred ➜ lambda_lift ➜ elim_hof ➜
      elim_rec ➜ intro_guards ➜ model_clean ➜ close {to_fo ➜ flatten {cvc4 ➜ id}}
    }
}
```

OCaml for translation pipeline

```
let pipe_common k =
  Step_tyinfer.pipe ~print:(!pp_typed_ || !pp_all_) @@@
  Step_conv_ty.pipe () @@@
  Tr.Skolem.pipe
    ~skolems_in_model:!skolems_in_model_
    ~print:(!pp_skolem_ || !pp_all_) ~check ~mode:`Sk_types @@@
  k
and pipe_mono_common k =
  Tr.Monomorphization.pipe
    ~always_mangle:false ~print:(!pp_mono_ || !pp_all_) ~check @@@
  Tr.Elim_infinite.pipe ~print:(!pp_elim_infinite || !pp_all_) ~check @@@
  Tr.ElimCopy.pipe ~print:(!pp_copy_ || !pp_all_) ~check @@@
  Tr.ElimMultipleEqns.pipe
    ~decode:(fun x->x) ~check
    ~print:(!pp_elim_multi_eqns || !pp_all_) @@@
  (if !enable_specialize_
   then Tr.Specialize.pipe ~print:(!pp_specialize_ || !pp_all_) ~check
   else Transform.nop ()) @@@
  k
and pipe_common_paradox_kodkod k =
  Tr.ElimData.Codata.pipe ~print:(!pp_elim_codata_ || !pp_all_) ~check @@@
  (if !enable_polarize_
   then Tr.Polarize.pipe ~print:(!pp_polarize_ || !pp_all_)
       ~check ~polarize_rec:!polarize_rec_
   else Transform.nop ()) @@@
  Tr.Unroll.pipe ~print:(!pp_unroll_ || !pp_all_) ~check @@@
  Tr.Skolem.pipe
    ~skolems_in_model:!skolems_in_model_
    ~print:(!pp_skolem_ || !pp_all_) ~mode:`Sk_all ~check @@@
  Tr.ElimIndPreds.pipe ~print:(!pp_elim_preds_ || !pp_all_)
    ~check ~mode:`Use_selectors @@@
  Tr.ElimData.Data.pipe ~print:(!pp_elim_data_ || !pp_all_) ~check @@@
  Tr.LambdaLift.pipe ~print:(!pp_lambda_lift_ || !pp_all_) ~check @@@
  Tr.Elim_HOF.pipe ~print:(!pp_elim_hof_ || !pp_all_) ~check @@@
  Tr.ElimRecursion.pipe ~print:(!pp_elim_recursion_ || !pp_all_) ~check @@@
  Tr.IntroGuards.pipe ~print:(!pp_intro_guards_ || !pp_all_) ~check @@@
  Tr.Elim_prop_args.pipe ~print:(!pp_elim_prop_args_ || !pp_all_) ~check @@@
  k
```

• • •

4. **Lean Forward**

Usable proofs and computations for number theorists

Future joint work with
Sander Dahmen, Gabriel Ebner, Johannes Hölzl,
Rob Lewis, Assia Mahboubi, Freek Wiedijk,
and many others

Vision

Prove modern theorems
(motivated by Sander Dahmen et al.'s research and interests)

Develop math libraries and automation
(e.g. basic algebraic number theory)

Develop tools, integrations
(e.g. Rob Lewis's Mathematica bridge, Nunchaku)

Develop Lean itself (C++)

Jasmin Christian Blanchette

So what are hammers (and counterexample generators) good for?

VU VRIJE UNIVERSITEIT AMSTERDAM