Automatic
Proofs and Refutations
in Isabelle/HOL

A Survey

Tobias Nipkow

Institut für Informatik
Technische Universität München

Isabelle

Isabelle

- Interactive theorem prover

Isabelle

- Interactive theorem prover
- Based on higher-order logic

Proof in Isabelle

Follows the *LCF-approach*:

The only way to derive new theorems is
by composing inference rules of the logic

Proof in Isabelle

Follows the *LCF-approach*:

The only way to derive new theorems is
by composing inference rules of the logic
(and previously proved theorems)

Proof in Isabelle

Follows the *LCF-approach*:

The only way to derive new theorems is
by composing inference rules of the logic
(and previously proved theorems)

$\Rightarrow$ All theorems are correct by construction!

Proof in Isabelle

Follows the *LCF-approach*:

The only way to derive new theorems is
by composing inference rules of the logic
(and previously proved theorems)

⇒ All theorems are correct by construction!

An architecture for complex proof procedures:

1. Produce some certificate
   (possibly summarizing a long search).

Proof in Isabelle

Follows the *LCF-approach*:

The only way to derive new theorems is
by composing inference rules of the logic
(and previously proved theorems)

$\Rightarrow$ All theorems are correct by construction!

An architecture for complex proof procedures:

1. Produce some certificate
(possibly summarizing a long search).
2. Translate the certificate into a theorem.

1. Automatic Proof
    - Internal Provers
    - External Provers
        - FO ATPs
        - SMT Solvers
        - SDP Solvers
2. Automatic Refutation
    - Nitpick
    - Quickcheck

1. Automatic Proof
   - Internal Provers
   - External Provers
     - FO ATPs
     - SMT Solvers
     - SDP Solvers
2. Automatic Refutation
   - Nitpick
   - Quickcheck

1. Automatic Proof
   - Internal Provers
   - External Provers
     - FO ATPs
     - SMT Solvers
     - SDP Solvers
2. Automatic Refutation
   - Nitpick
   - Quickcheck

The workhorses

The provers:

- `simp`, `auto`, `force`:

The workhorses

The provers:

- `simp`, `auto`, `force`:
  rewriting, a bit of arithmetic, more and more logic

The workhorses

The provers:

- `simp`, `auto`, `force`:
  rewriting, a bit of arithmetic, more and more logic
  [rewriting interleaved with tableau]

The workhorses

The provers:

- `simp`, `auto`, `force`:
  rewriting, a bit of arithmetic, more and more logic
  [rewriting interleaved with tableau]
- `blast`:

The workhorses

The provers:

- `simp`, `auto`, `force`:
  rewriting, a bit of arithmetic, more and more logic
  [rewriting interleaved with tableau]
- `blast`:
  logic, sets and relations, almost no $=$, no arithmetic.

The workhorses

The provers:

- `simp`, `auto`, `force`:
  rewriting, a bit of arithmetic, more and more logic
  [rewriting interleaved with tableau]
- `blast`:
  logic, sets and relations, almost no $=$, no arithmetic.
  [tableau]

The workhorses

The provers:

- `simp`, `auto`, `force`:
  rewriting, a bit of arithmetic, more and more logic
  [rewriting interleaved with tableau]
- `blast`:
  logic, sets and relations, almost no $=$, no arithmetic.
  [tableau]

The user perspective:

- Frequently fast and effective

The workhorses

The provers:

- `simp`, `auto`, `force`:
  rewriting, a bit of arithmetic, more and more logic
  [rewriting interleaved with tableau]
- `blast`:
  logic, sets and relations, almost no $=$, no arithmetic.
  [tableau]

The user perspective:

- Frequently fast and effective
- Sometimes annoyingly incomplete

Metis

by Joe Hurd for HOL4

- Ordered resolution

Metis

by Joe Hurd for HOL4

- Ordered resolution
- Written in SML

Metis

by Joe Hurd for HOL4

- Ordered resolution
- Written in SML
- Performs in CASC (lower third)

Metis

by Joe Hurd for HOL4

- Ordered resolution
- Written in SML
- Performs in CASC (lower third)
- Generates resolution proof

Metis

by Joe Hurd for HOL4

- Ordered resolution
- Written in SML
- Performs in CASC (lower third)
- Generates resolution proof
- Isabelle theorem generated in a second step

Arithmetic and Algebra

- `arith`: linear real arithmetic & Presburger arithmetic

Arithmetic and Algebra

- `arith`: linear real arithmetic & Presburger arithmetic
- `algebra`: Gröbner basis

1. Automatic Proof
   - Internal Provers
   - External Provers
     - FO ATPs
     - SMT Solvers
     - SDP Solvers
2. Automatic Refutation
   - Nitpick
   - Quickcheck

1. Automatic Proof
    - Internal Provers
    - External Provers
        - FO ATPs
        - SMT Solvers
        - SDP Solvers
2. Automatic Refutation
    - Nitpick
    - Quickcheck

Sledgehammer

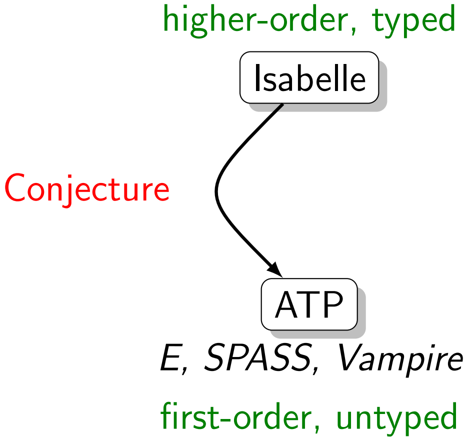
by Paulson, Meng, Susanto, Quigley (at Cambridge)
Wenzel, Immler, Meyer, Blanchette (at Munich)

How  works

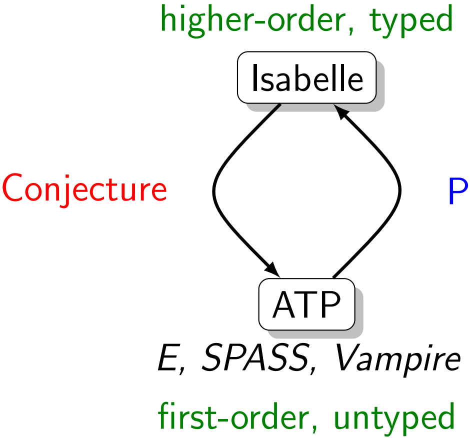
higher-order, typed

Isabelle

How  works


higher-order, typed
Isabelle
Conjecture
ATP
E, SPASS, Vampire
first-order, untyped

How  works


higher-order, typed
Isabelle
Conjecture
Proof
ATP
E, SPASS, Vampire
first-order, untyped

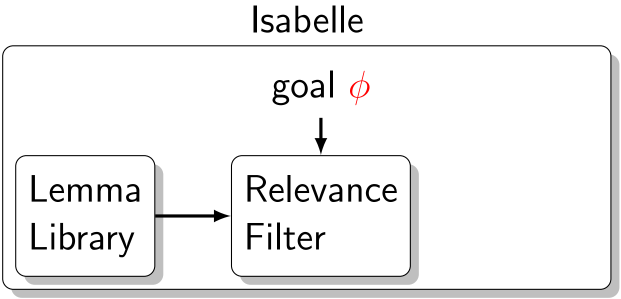
How  works

Isabelle

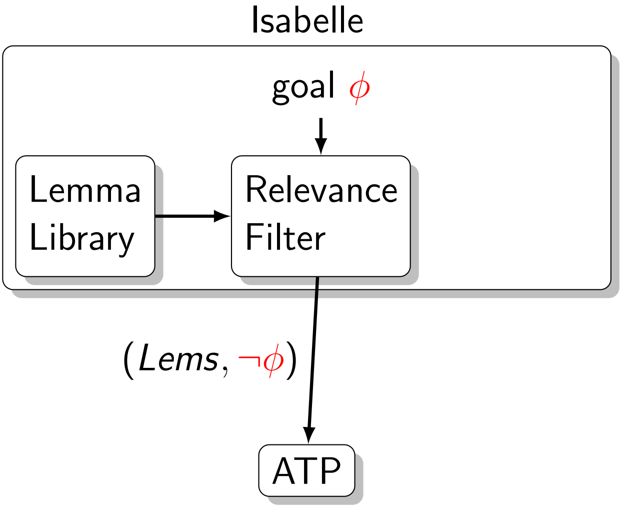
goal ϕ

How  works

How  works

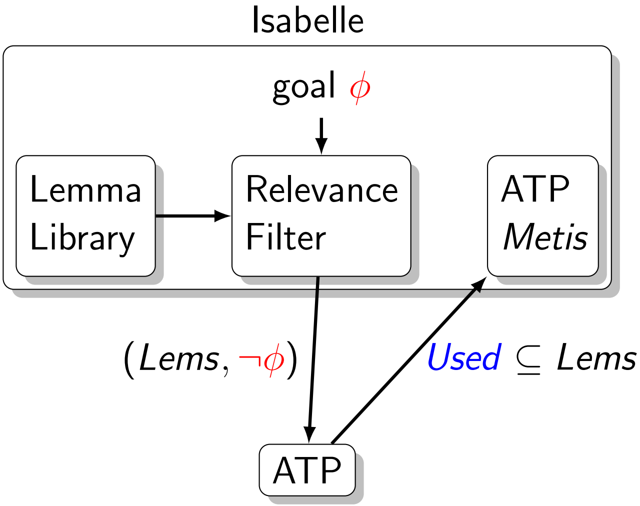


How  works

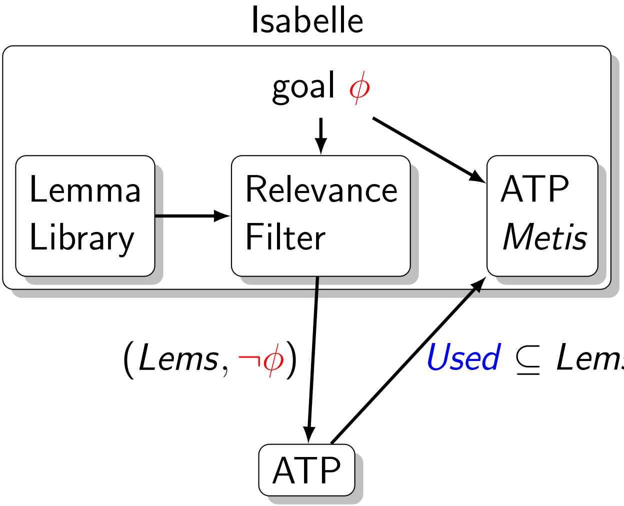


Isabelle
goal ϕ
Lemma Library
Relevance Filter
$(Lems, \neg\phi)$
ATP

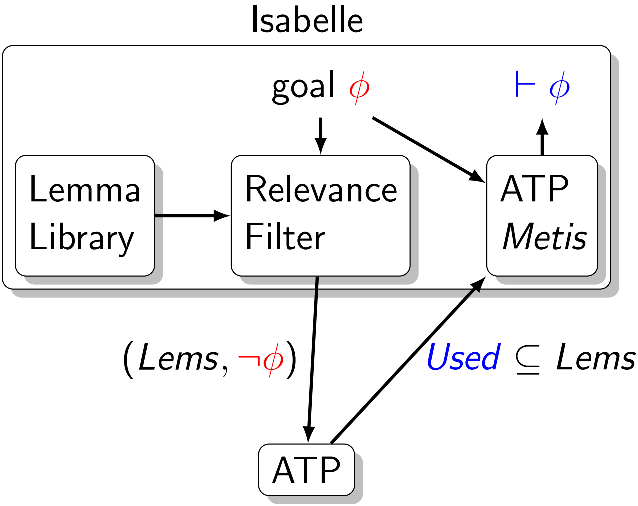
How  works


Isabelle
goal ϕ
Lemma Library
Relevance Filter
ATP *Metis*
$(Lems, \neg\phi)$
$Used \subseteq Lems$
ATP

How  works

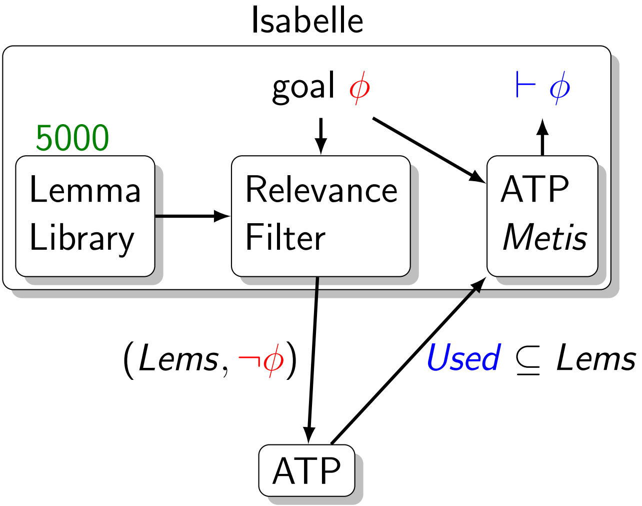

Isabelle
goal ϕ
Lemma Library
Relevance Filter
ATP Metis
$(Lems, \neg\phi)$
$Used \subseteq Lems$
ATP

How  works

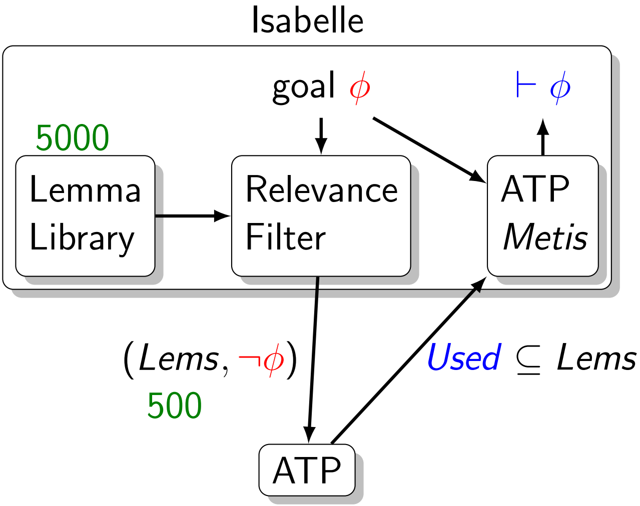


Isabelle
goal ϕ
$\vdash \phi$
Lemma Library
Relevance Filter
ATP *Metis*
$(Lems, \neg\phi)$
Used $\subseteq$ *Lems*
ATP

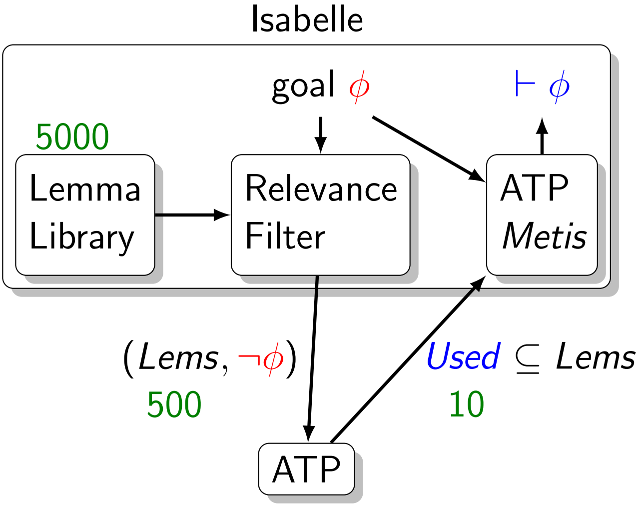
How  works


Isabelle
goal ϕ
$\vdash \phi$
5000
Lemma Library
Relevance Filter
ATP *Metis*
$(Lems, \neg\phi)$
$Used \subseteq Lems$
ATP

How  works


Isabelle
goal ϕ
$\vdash \phi$
5000
Lemma Library
Relevance Filter
ATP Metis
$(Lems, \neg\phi)$
500
Used $\subseteq$ Lems
ATP

How  works


Isabelle
goal ϕ
$\vdash \phi$
5000
Lemma Library
Relevance Filter
ATP Metis
$(Lems, \neg\phi)$
500
Used $\subseteq$ Lems
10
ATP

External ATPs act as relevance filter for Metis

Sledgehammer: proofs

- Short, cryptic, expensive to reconstruct:

  lemma $f\ xs \neq Suc\ 0$

Sledgehammer: proofs

- Short, cryptic, expensive to reconstruct:

  lemma *f xs* ≠ *Suc* 0
  by (*metis f.simps less_Suc_eq_0_disj list.exhaust less_irrefl Suc_not_Zero*)

Sledgehammer: proofs

- Short, cryptic, expensive to reconstruct:

  lemma $f\ xs \neq Suc\ 0$
  by (*metis f.simps less_Suc_eq_0_disj list.exhaust less_irrefl Suc_not_Zero*)

- Long, readable, easy to reconstruct:

  lemma $f\ xs \neq Suc\ 0$

Sledgehammer: proofs

- Short, cryptic, expensive to reconstruct:

lemma *f xs* $\neq$ *Suc* 0
by (*metis f.simps less_Suc_eq_0_disj list.exhaust less_irrefl*
    Suc_not_Zero)

- Long, readable, easy to reconstruct:

lemma *f xs* $\neq$ *Suc* 0
proof $-$
  have *F1*: $\forall x_1.\ Suc\ 0 < f\ x_1 \vee x_1 = []$
    by (*metis f.simps*(2) *less_Suc_eq_0_disj list.exhaust*)
  { **assume** *xs* $\neq$ []
    hence *f xs* $\neq$ *Suc* 0 **by** (*metis F1 less_irrefl*) }
  thus *f xs* $\neq$ *Suc* 0 **by** (*metis f.simps*(1) *Suc_not_Zero*)
qed

Sledgehammer: proofs

- Short, cryptic, expensive to reconstruct:

lemma *f xs* $\neq$ *Suc* 0
by (*metis f.simps less_Suc_eq_0_disj list.exhaust less_irrefl*
Suc_not_Zero)

- Long, readable, easy to reconstruct:

lemma *f xs* $\neq$ *Suc* 0
proof –
have *F1*: $\forall x_1.\ Suc\ 0 < f\ x_1 \lor x_1 = []$
by (*metis f.simps*(2) *less_Suc_eq_0_disj list.exhaust*)
{ **assume** *xs* $\neq$ []
hence *f xs* $\neq$ *Suc* 0 **by** (*metis F1 less_irrefl*) }
thus *f xs* $\neq$ *Suc* 0 **by** (*metis f.simps*(1) *Suc_not_Zero*)
qed

Work in progress!

Sledgehammer: empirical evaluation

Based on 1200 goals from diverse theories covering

- arithmetic
- inductive datatypes
- recursive functions
- inductive definitions
- set theory

Sledgehammer: some key findings

- 45% of all goals (not lemmas!) can be proved automatically

Sledgehammer: some key findings

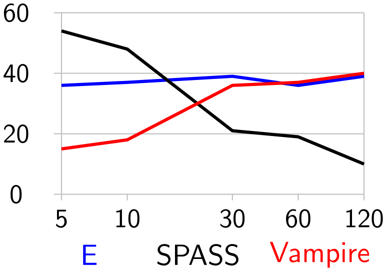
- 45% of all goals (not lemmas!) can be proved automatically
- 33% of all non-trivial goals can be proved automatically

Sledgehammer: some key findings

- 45% of all goals (not lemmas!) can be proved automatically
- 33% of all non-trivial goals can be proved automatically
- 3 ATPs for 5 secs $\geq$ 1 ATP for 120 secs

Sledgehammer: some key findings

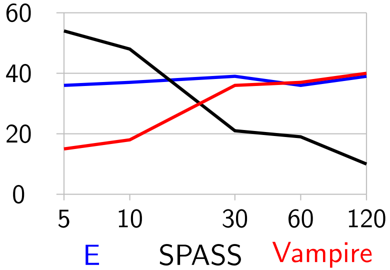
- 45% of all goals (not lemmas!) can be proved automatically
- 33% of all non-trivial goals can be proved automatically
- 3 ATPs for 5 secs $\geq$ 1 ATP for 120 secs



number of goals only a particular prover can prove

Sledgehammer: some key findings

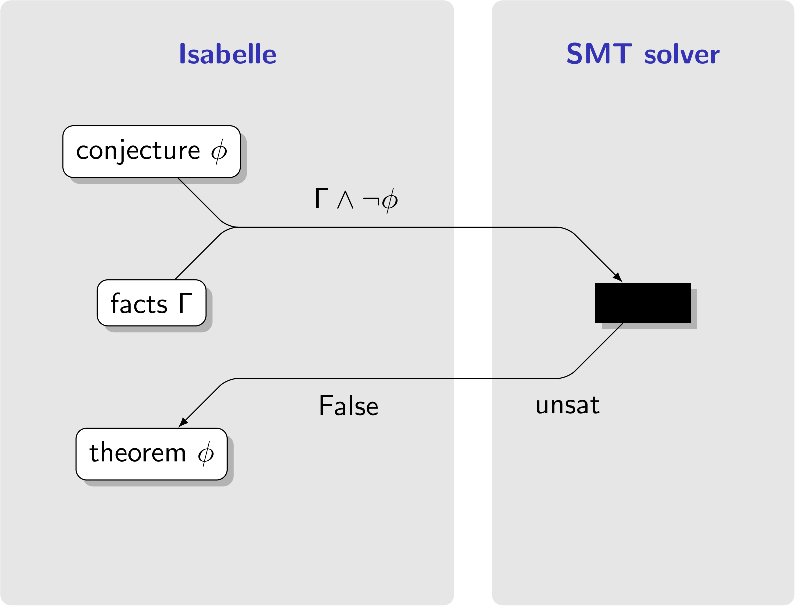
- 45% of all goals (not lemmas!) can be proved automatically
- 33% of all non-trivial goals can be proved automatically
- 3 ATPs for 5 secs $\geq$ 1 ATP for 120 secs



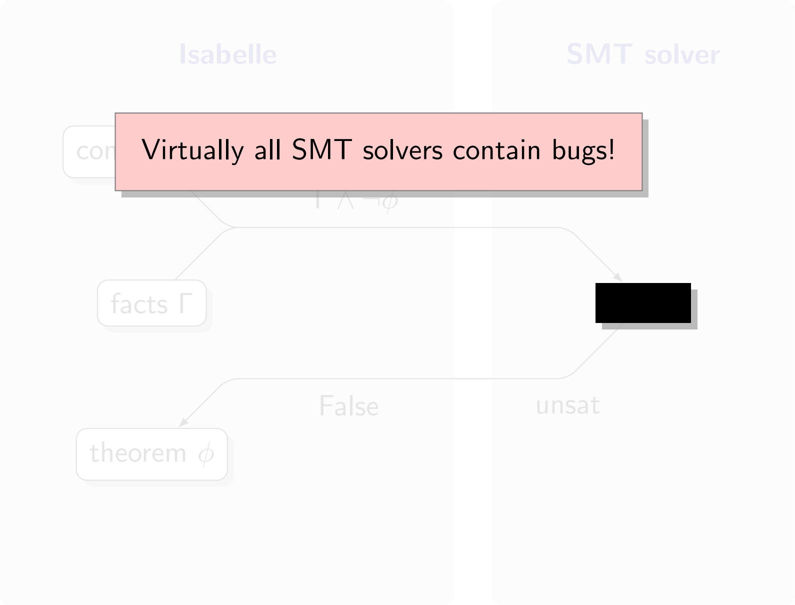
number of goals only a particular prover can prove

[Böhme, Nipkow, IJCAR 2010]

Integration of SMT Solvers



Integration of SMT Solvers



Virtually all SMT solvers contain bugs!

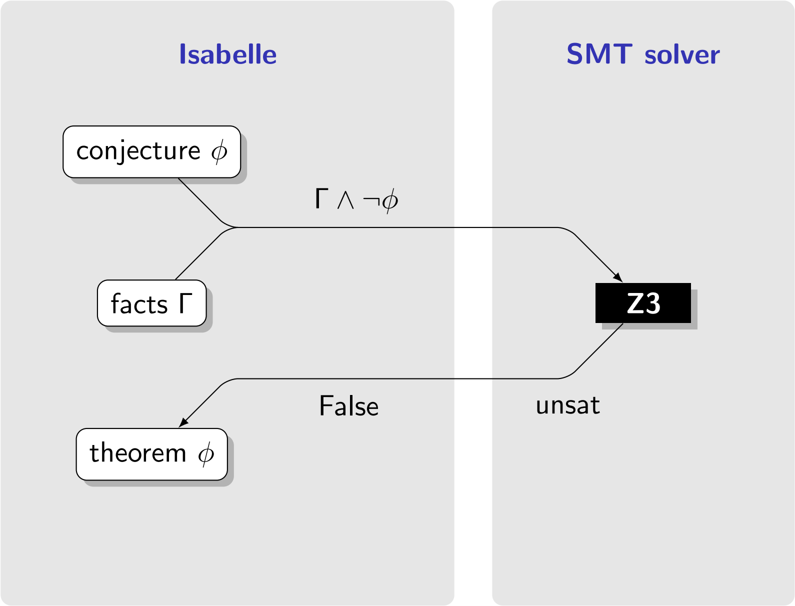
Integration of SMT Solvers

Virtually all SMT solvers contain bugs!

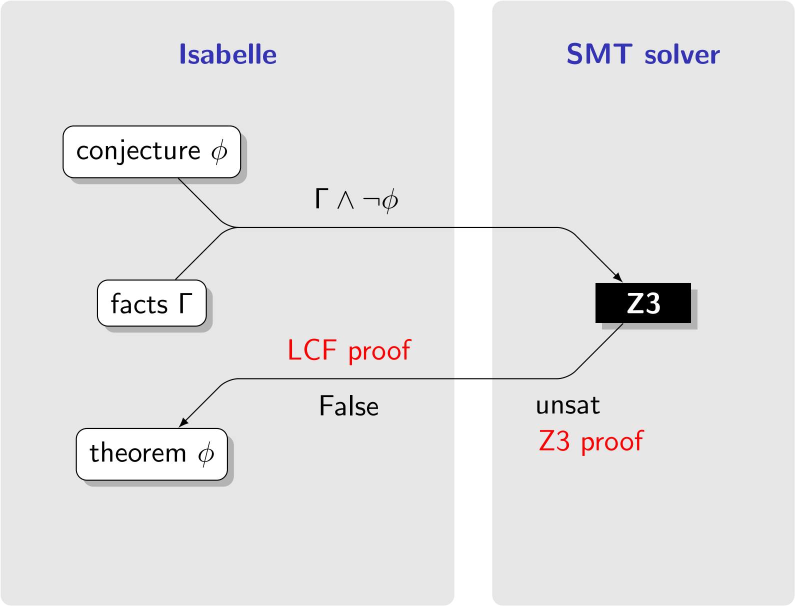
- leading SMT solver
- developed by Microsoft Research
- supports proofs

Z3

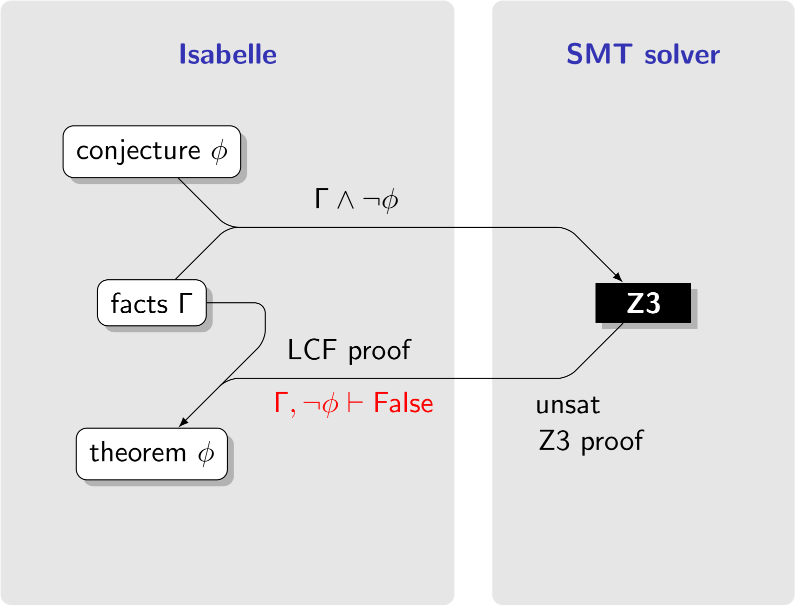
Integration of SMT Solvers



Integration of SMT Solvers



Integration of SMT Solvers



Z3 Proof Format

Z3's proofs are conceptually simple:

- natural deduction
- (only) 34 inference rules
- theory reasoning: only two generic inference rules

Z3 Proof Format

Z3's proofs are conceptually simple:

- natural deduction
- (only) 34 inference rules
- theory reasoning: only two generic inference rules

Th-Lemma: inconsistency of theory atoms

- no hint towards kind of theory
- no further proof explanation

Z3 Proof Format

Z3's proofs are conceptually simple:

- natural deduction
- (only) 34 inference rules
- theory reasoning: only two generic inference rules

TH-LEMMA: inconsistency of theory atoms

- no hint towards kind of theory
- no further proof explanation

REWRITE: propositional/theory-related equivalences/equalities

- nearly unspecified . . .

Z3 Proof Format

Z3's proofs are conceptually simple:

- natural deduction
- (only) 34 inference rules
- theory reasoning: only two generic inference rules

Th-Lemma: inconsistency of theory atoms

- no hint towards kind of theory
- no further proof explanation

Rewrite: propositional/theory-related equivalences/equalities

- nearly unspecified ...

Theory rules are lacking valuable information!

Proof Reconstruction for Z3

Proof reconstruction:

- one Isabelle proof method for each Z3 inference rule
- depth-first traversal through Z3 proof graph

Proof Reconstruction for Z3

Proof reconstruction:

- one Isabelle proof method for each Z3 inference rule
- depth-first traversal through Z3 proof graph

Bottleneck: theory reasoning

- TH-LEMMA: expensive proof search
- REWRITE: try a bunch of different specific proof methods and internal provers

Proof Reconstruction for Z3

Proof reconstruction:

- one Isabelle proof method for each Z3 inference rule
- depth-first traversal through Z3 proof graph

Bottleneck: theory reasoning

- TH-LEMMA: expensive proof search
- REWRITE: try a bunch of different specific proof methods and internal provers

[Böhme, Weber, ITP2010]

Evaluation

SMT-LIB benchmarks:

Logic	Z3			Isabelle		Rates	
	#	Med. Time	Med. Size	#	Med. Time	Succ.	Time-out
AUFLIA+p	187	0.03 s	5 KB	187	0.06 s	100%	0%
AUFLIA−p	192	0.04 s	4 KB	190	0.06 s	98%	0%
AUFLIRA	189	0.02 s	16 KB	144	0.04 s	76%	0%
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Total	1273	3.66 s	13 MB	962	11.31 s	75%	19%

Evaluation

SMT-LIB benchmarks:

Logic	Z3			Isabelle		Rates	
	#	Med. Time	Med. Size	#	Med. Time	Succ.	Time-out
AUFLIA+p	187	0.03 s	5 KB	187	0.06 s	100%	0%
AUFLIA−p	192	0.04 s	4 KB	190	0.06 s	98%	0%
AUFLIRA	189	0.02 s	16 KB	144	0.04 s	76%	0%
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Total	1273	3.66 s	13 MB	962	11.31 s	75%	19%

Evaluation

SMT-LIB benchmarks:

Logic	Z3			Isabelle		Rates	
	#	Med. Time	Med. Size	#	Med. Time	Succ.	Time-out
AUFLIA+p	187	0.03 s	5 KB	187	0.06 s	100%	0%
AUFLIA−p	192	0.04 s	4 KB	190	0.06 s	98%	0%
AUFLIRA	189	0.02 s	16 KB	144	0.04 s	76%	0%
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Total	1273	3.66 s	13 MB	962	11.31 s	75%	19%

Profiling:

- bottleneck: theory reasoning requires expensive proof search
- 50% of the runtime is spent on 15% of all Z3 proof steps

1. Automatic Proof
   - Internal Provers
   - External Provers
     - FO ATPs
     - SMT Solvers
     - SDP Solvers
2. Automatic Refutation
   - Nitpick
   - Quickcheck

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

- To prove $p(x_1, \ldots, x_n) \geq 0$, express p as a sum of squares

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

- To prove $p(x_1, \ldots, x_n) \geq 0$, express p as a sum of squares
- Generalized to boolean combinations of $p \geq q$ and $p = q$

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

- To prove $p(x_1, \ldots, x_n) \geq 0$, express p as a sum of squares
- Generalized to boolean combinations of $p \geq q$ and $p = q$
- The SOS decomposition is found with the help of an external *SDP solver*

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

- To prove $p(x_1, \ldots, x_n) \geq 0$, express p as a sum of squares
- Generalized to boolean combinations of $p \geq q$ and $p = q$
- The SOS decomposition is found with the help of an external *SDP solver*
- Incomplete in theory

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

- To prove $p(x_1, \ldots, x_n) \geq 0$, express p as a sum of squares
- Generalized to boolean combinations of $p \geq q$ and $p = q$
- The SOS decomposition is found with the help of an external *SDP solver*
- Incomplete in theory
- Works well in practice

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

- To prove $p(x_1, \dots, x_n) \geq 0$, express p as a sum of squares
- Generalized to boolean combinations of $p \geq q$ and $p = q$
- The SOS decomposition is found with the help of an external *SDP solver*
- Incomplete in theory
- Works well in practice

The Isabelle implementation:

- Ported from HOL Light

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

- To prove $p(x_1, \ldots, x_n) \geq 0$, express p as a sum of squares
- Generalized to boolean combinations of $p \geq q$ and $p = q$
- The SOS decomposition is found with the help of an external *SDP solver*
- Incomplete in theory
- Works well in practice

The Isabelle implementation:

- Ported from HOL Light
- Generates certificates that allow proof replay w/o SDP solver

Non-linear arithmetic

The sum-of-squares (SOS) method (by John Harrison):

- To prove $p(x_1, \ldots, x_n) \geq 0$, express p as a sum of squares
- Generalized to boolean combinations of $p \geq q$ and $p = q$
- The SOS decomposition is found with the help of an external *SDP solver*
- Incomplete in theory
- Works well in practice

The Isabelle implementation:

- Ported from HOL Light
- Generates certificates that allow proof replay w/o SDP solver

Example: $x^2 + y^2 + z^2 = 1 \Longrightarrow (x + y + z)^2 \leq 3$

1. Automatic Proof
   - Internal Provers
   - External Provers
     - FO ATPs
     - SMT Solvers
     - SDP Solvers
2. Automatic Refutation
   - Nitpick
   - Quickcheck

Counterexample Generation: Motivation

Why counterexamples are important

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of “theorems” contain errors

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of “theorems” contain errors
- Finding errors by failed proof attempts is expensive

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of “theorems” contain errors
- Finding errors by failed proof attempts is expensive
- Counterexample generation can help to find errors quickly

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of "theorems" contain errors
- Finding errors by failed proof attempts is expensive
- Counterexample generation can help to find errors quickly

Approaches for counterexample generation

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of “theorems” contain errors
- Finding errors by failed proof attempts is expensive
- Counterexample generation can help to find errors quickly

Approaches for counterexample generation

Nitpick search for finite countermodels of formula using external tools (SAT solvers)

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of "theorems" contain errors
- Finding errors by failed proof attempts is expensive
- Counterexample generation can help to find errors quickly

Approaches for counterexample generation

Nitpick search for finite countermodels of formula using external tools (SAT solvers)

- Covers most of HOL, including non-executable constructs (e.g. quantifiers)

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of "theorems" contain errors
- Finding errors by failed proof attempts is expensive
- Counterexample generation can help to find errors quickly

Approaches for counterexample generation

Nitpick search for finite countermodels of formula using external tools (SAT solvers)

- Covers most of HOL, including non-executable constructs (e.g. quantifiers)
- Slow for complex data structures

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of "theorems" contain errors
- Finding errors by failed proof attempts is expensive
- Counterexample generation can help to find errors quickly

Approaches for counterexample generation

Nitpick search for finite countermodels of formula using external tools (SAT solvers)

- Covers most of HOL, including non-executable constructs (e.g. quantifiers)
- Slow for complex data structures

Quickcheck evaluate formula on random values for free variables using code generator

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of "theorems" contain errors
- Finding errors by failed proof attempts is expensive
- Counterexample generation can help to find errors quickly

Approaches for counterexample generation

Nitpick search for finite countermodels of formula using external tools (SAT solvers)

- Covers most of HOL, including non-executable constructs (e.g. quantifiers)
- Slow for complex data structures

Quickcheck evaluate formula on random values for free variables using code generator

- Fast

Counterexample Generation: Motivation

Why counterexamples are important

- Complex specifications are hard to get right
- Many statements of "theorems" contain errors
- Finding errors by failed proof attempts is expensive
- Counterexample generation can help to find errors quickly

Approaches for counterexample generation

Nitpick: search for finite countermodels of formula using external tools (SAT solvers)

- Covers most of HOL, including non-executable constructs (e.g. quantifiers)
- Slow for complex data structures

Quickcheck: evaluate formula on random values for free variables using code generator

- Fast
- Restricted to executable fragment of HOL

Nitpick: How It Works

Nitpick: How It Works

Nitpick:

- converts HOL formula to first-order relational logic (FORL)

Nitpick: How It Works

Nitpick:

- converts HOL formula to first-order relational logic (FORL)
- invokes the SAT-based Kodkod model finder (Alloy's backend) on FORL formula

Nitpick: How It Works

Nitpick:

- converts HOL formula to first-order relational logic (FORL)
- invokes the SAT-based Kodkod model finder (Alloy's backend) on FORL formula
- handles HOL's definitional principles specially:
  - (co)inductive predicates and datatypes
  - (co)recursive functions

Nitpick: How It Works

Nitpick:

- converts HOL formula to first-order relational logic (FORL)
- invokes the SAT-based Kodkod model finder (Alloy's backend) on FORL formula
- handles HOL's definitional principles specially:
  - (co)inductive predicates and datatypes
  - (co)recursive functions
- optimizes common higher-order idioms

Nitpick: How It Works

Nitpick:

- converts HOL formula to first-order relational logic (FORL)
- invokes the SAT-based Kodkod model finder (Alloy's backend) on FORL formula
- handles HOL's definitional principles specially:
  - (co)inductive predicates and datatypes
  - (co)recursive functions
- optimizes common higher-order idioms

[Blanchette, Nipkow, ITP-10]

Nitpick: Small Examples

1. $(A \cup B)^{+} = A^{+} \cup B^{+}$

Nitpick: Small Examples

1. $(A \cup B)^+ = A^+ \cup B^+$
   $A = \{(2,1)\} \quad B = \{(1,2)\}$

Nitpick: Small Examples

1. $(A \cup B)^+ = A^+ \cup B^+$
   $A = \{(2,1)\} \quad B = \{(1,2)\}$

2. $xs \mathbin{@} ys = xs \longleftrightarrow ys = []$

Nitpick: Small Examples

1. $(A \cup B)^+ = A^+ \cup B^+$
   $A = \{(2, 1)\} \quad B = \{(1, 2)\}$

2. $xs \mathbin{@} ys = xs \longleftrightarrow ys = []$ (for coinductive lists)

Nitpick: Small Examples

1. $(A \cup B)^+ = A^+ \cup B^+$
   $A = \{(2, 1)\} \quad B = \{(1, 2)\}$

2. $xs \mathbin{@} ys = xs \longleftrightarrow ys = []$ (for coinductive lists)
   $xs = ys = [1, 1, \ldots]$

Nitpick: Empirical Evaluation

Mutation testing:

Nitpick: Empirical Evaluation

Mutation testing:

On average, Nitpick falsifies ≈42% of all mutants

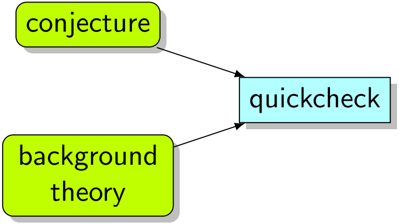
1. Automatic Proof
   - Internal Provers
   - External Provers
     - FO ATPs
     - SMT Solvers
     - SDP Solvers
2. Automatic Refutation
   - Nitpick
   - Quickcheck

Quickcheck

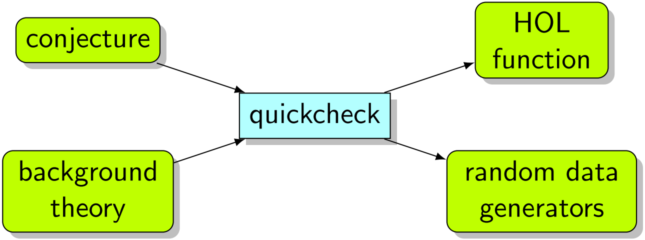
conjecture

background theory

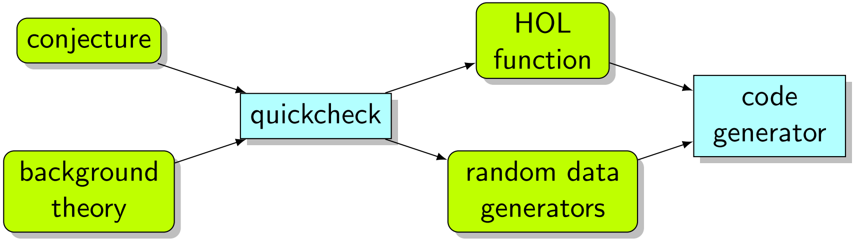
Quickcheck



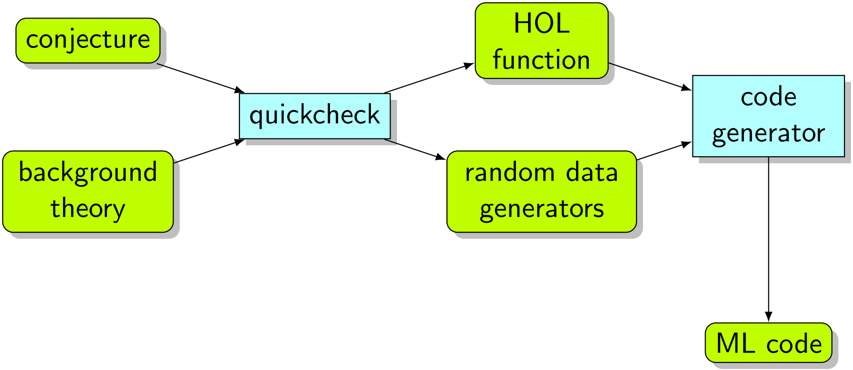
Quickcheck


conjecture
background theory
quickcheck
HOL function
random data generators

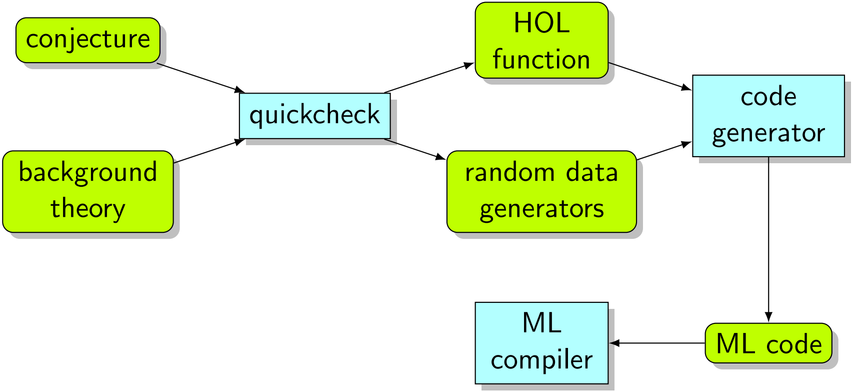
Quickcheck


conjecture
background theory
quickcheck
HOL function
random data generators
code generator

Quickcheck

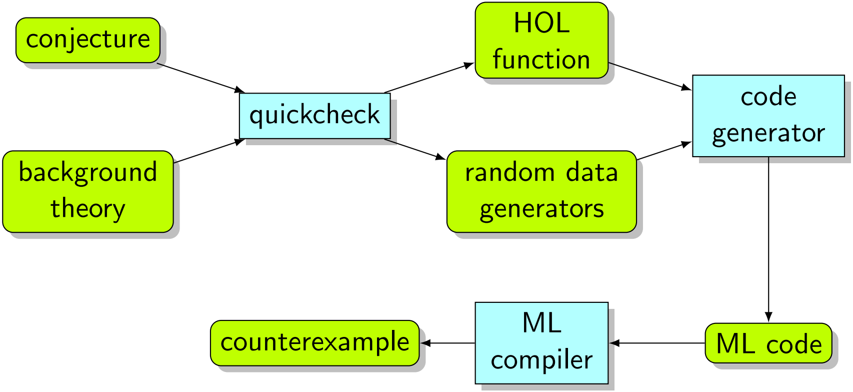

conjecture
background theory
quickcheck
HOL function
random data generators
code generator
ML code

Quickcheck



conjecture
background theory
quickcheck
HOL function
random data generators
code generator
ML code
ML compiler

Quickcheck



conjecture
background theory
quickcheck
HOL function
random data generators
code generator
ML code
ML compiler
counterexample

Current and future work

Focus: Sledgehammer

Current and future work

Focus: Sledgehammer

- Can we give the ATPs more information?

Current and future work

Focus: Sledgehammer

- Can we give the ATPs more information?
- Proof replay in Isabelle

Current and future work

Focus: Sledgehammer

- Can we give the ATPs more information?
- Proof replay in Isabelle
- Improved treatment of HO problems

Current and future work

Focus: Sledgehammer

- Can we give the ATPs more information?
- Proof replay in Isabelle
- Improved treatment of HO problems

What we would like:

Current and future work

Focus: Sledgehammer

- Can we give the ATPs more information?
- Proof replay in Isabelle
- Improved treatment of HO problems

What we would like:

- bigger, better, faster, more ATPs

Current and future work

Focus: Sledgehammer

- Can we give the ATPs more information?
- Proof replay in Isabelle
- Improved treatment of HO problems

What we would like:

- bigger, better, faster, more ATPs
- bigger, better, faster, more SMT solvers

Current and future work

Focus: Sledgehammer

- Can we give the ATPs more information?
- Proof replay in Isabelle
- Improved treatment of HO problems

What we would like:

- bigger, better, faster, more ATPs
- bigger, better, faster, more SMT solvers
- bigger, better, faster, more SAT solvers