Testing a Saturation-Based Theorem Prover: Experiences and Challenges

Giles Reger[1], Martin Suda[2], and Andrei Voronkov[1,2]

[1]School of Computer Science, University of Manchester, UK
[2]TU Wien, Vienna, Austria

TAP 2017 – Marburg, July 19, 2017

Introduction

First-order Automatic Theorem Proving:

- a well-established discipline of automated deduction
- main approach: refutational, saturation-based proving
- example systems: E, SPASS, Vampire

Introduction

First-order Automatic Theorem Proving:

- a well-established discipline of automated deduction
- main approach: refutational, saturation-based proving
- example systems: E, SPASS, Vampire

Often used in larger projects and systems as black boxes

- e.g., program verification, static analysis, interpolation, ...

Introduction

First-order Automatic Theorem Proving:

- a well-established discipline of automated deduction
- main approach: refutational, saturation-based proving
- example systems: E, SPASS, Vampire

Often used in larger projects and systems as black boxes

- e.g., program verification, static analysis, interpolation, ...

➥ Importance of ensuring correctness

Introduction

First-order Automatic Theorem Proving:

- a well-established discipline of automated deduction
- main approach: refutational, saturation-based proving
- example systems: E, SPASS, Vampire

Often used in larger projects and systems as black boxes

- e.g., program verification, static analysis, interpolation, ...

➥ Importance of ensuring correctness

How are we doing?

Introduction

First-order Automatic Theorem Proving:

- a well-established discipline of automated deduction
- main approach: refutational, saturation-based proving
- example systems: E, SPASS, Vampire

Often used in larger projects and systems as black boxes

- e.g., program verification, static analysis, interpolation, ...

➥ Importance of ensuring correctness

How are we doing?

- CASC competition: preliminary period for testing soundness

Introduction

First-order Automatic Theorem Proving:

- a well-established discipline of automated deduction
- main approach: refutational, saturation-based proving
- example systems: E, SPASS, Vampire

Often used in larger projects and systems as black boxes

- e.g., program verification, static analysis, interpolation, ...

➥ Importance of ensuring correctness

How are we doing?

- CASC competition: preliminary period for testing soundness
- SMT-COMP 2016: 79 answers classified as incorrect

Our Prover

Vampire

- Automatic Theorem Prover for first-order logic and theories

Our Prover

Vampire

- Automatic Theorem Prover for first-order logic and theories
- regular winner of the main divisions of the CASC competition

- since 2016, also a successful participant of SMT-COMP

Our Prover

Vampire

- Automatic Theorem Prover for first-order logic and theories
- regular winner of the main divisions of the CASC competition

- since 2016, also a successful participant of SMT-COMP

Quite complex piece of software (≈194000 lines of C++)

➥ easy to introduce incorrectness when adding a new feature

Outline

Theorem proving basics

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

Theorem proving basics

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

1. Negate F (to seek a refutation):

$$\neg F \quad := \quad Axiom_1 \wedge \ldots \wedge Axiom_n \wedge \neg Conjecture$$

Theorem proving basics

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

1. Negate F (to seek a refutation):

$$\neg F \quad := \quad Axiom_1 \wedge \ldots \wedge Axiom_n \wedge \neg Conjecture$$

2. Preprocess and transform $\neg F$ to a normal form

$$\mathcal{S} \quad := \quad \{C_1, \ldots, C_n\}$$

Theorem proving basics

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

1. Negate F (to seek a refutation):

$$\neg F \quad := \quad Axiom_1 \wedge \ldots \wedge Axiom_n \wedge \neg Conjecture$$

2. Preprocess and transform $\neg F$ to a normal form

$$\mathcal{S} \quad := \quad \{C_1, \ldots, C_n\}$$

3. saturate $\mathcal{S}$ with respect to an inference system $\mathcal{I}$

Theorem proving basics

Standard form of the input:

$$F \quad := \quad (Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$$

1. Negate F (to seek a refutation):

$$\neg F \quad := \quad Axiom_1 \wedge \ldots \wedge Axiom_n \wedge \neg Conjecture$$

2. Preprocess and transform $\neg F$ to a normal form

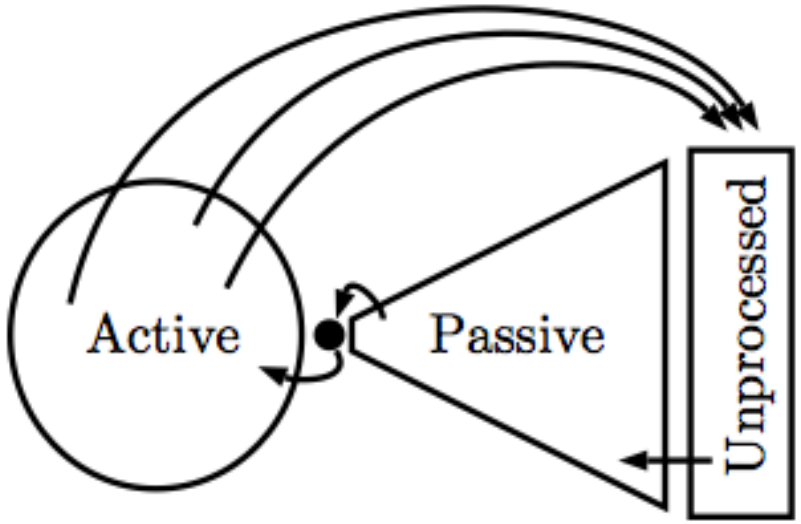
$$\mathcal{S} \quad := \quad \{C_1, \ldots, C_n\}$$

3. saturate $\mathcal{S}$ with respect to an inference system $\mathcal{I}$

Example inference rule: $\dfrac{C_1 \vee P \quad C_2 \vee \neg P}{C_1 \vee C_2}$

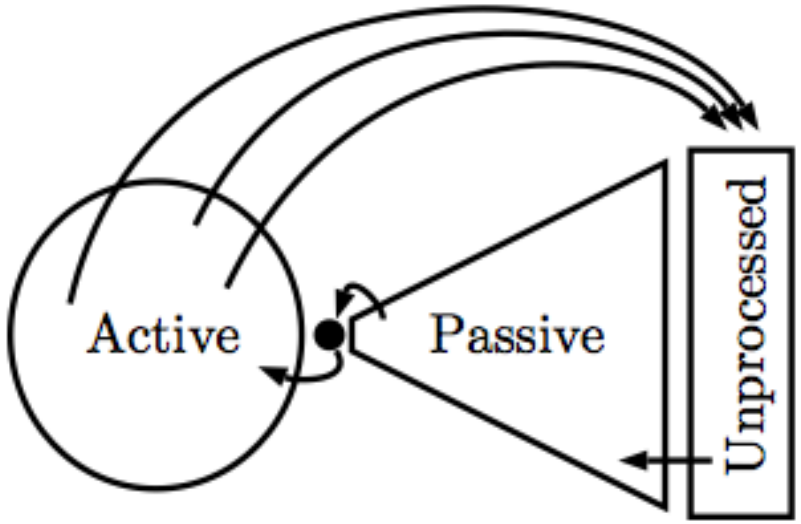
The Saturation Process

Saturation = fixed-point (closure) computation



The Saturation Process

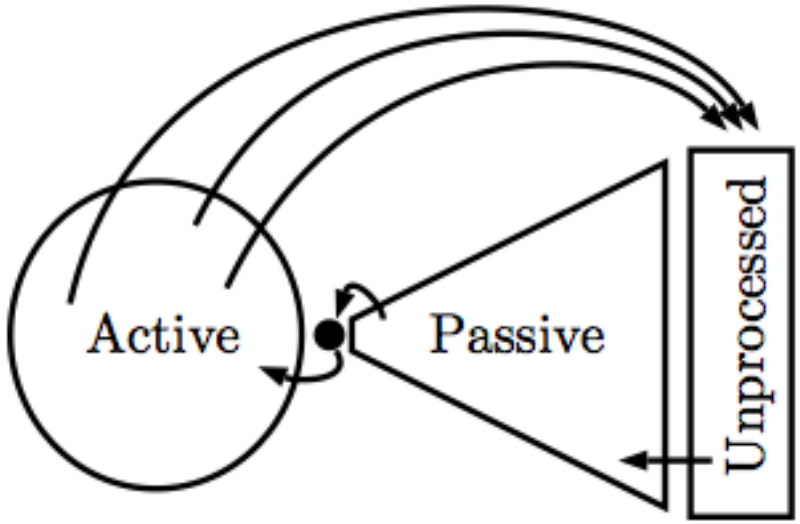
Saturation = fixed-point (closure) computation



Does the final set $\mathcal{S}$ contain *false*?

The Saturation Process

Saturation = fixed-point (closure) computation

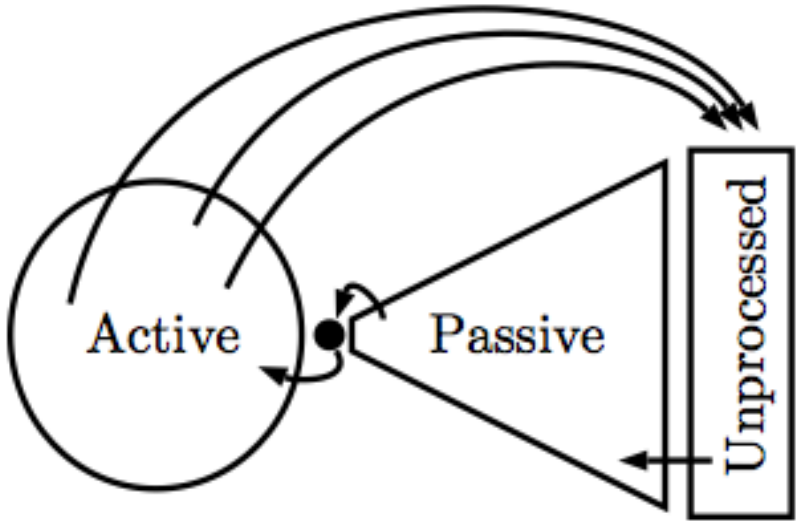


Does the final set $\mathcal{S}$ contain *false*?

Basic properties:

The Saturation Process

Saturation = fixed-point (closure) computation



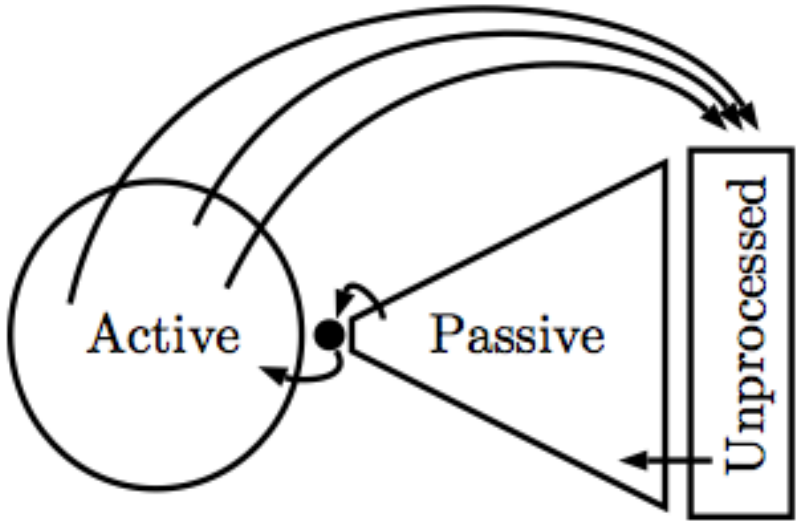
Does the final set $\mathcal{S}$ contain *false*?

Basic properties:

- explosive in nature

The Saturation Process

Saturation = fixed-point (closure) computation



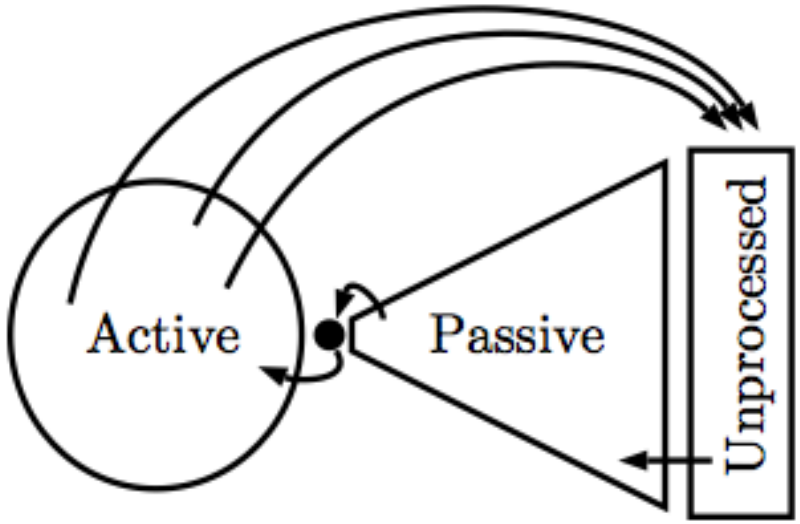
Does the final set $\mathcal{S}$ contain *false*?

Basic properties:

- explosive in nature
- may not terminate

The Saturation Process

Saturation = fixed-point (closure) computation



Does the final set $\mathcal{S}$ contain *false*?

Basic properties:

- explosive in nature
- may not terminate
- various tricks to mitigate the explosion

Possible Answers:

- `Theorem` (together with a proof)
  - if the input F is logically valid

Possible Answers:

- `Theorem` (together with a proof)
  - if the input F is logically valid
- `Non-theorem`
  - if F is invalid (there is a counter-example)

Possible Answers:

- `Theorem` (together with a proof)
  - if the input F is logically valid
- `Non-theorem`
  - if F is invalid (there is a counter-example)
  - relies on a completeness argument

Possible Answers:

- `Theorem` (together with a proof)
  - if the input F is logically valid
- `Non-theorem`
  - if F is invalid (there is a counter-example)
  - relies on a completeness argument
- `Unknown`

Possible Answers:

- `Theorem` (together with a proof)
  - if the input F is logically valid
- `Non-theorem`
  - if F is invalid (there is a counter-example)
  - relies on a completeness argument
- `Unknown`
  1. time limit / memory limit

Possible Answers:

- `Theorem` (together with a proof)
  - if the input F is logically valid
- `Non-theorem`
  - if F is invalid (there is a counter-example)
  - relies on a completeness argument
- `Unknown`
  1. time limit / memory limit
  2. incomplete strategy failed

Different Ways of Being Incorrect

unsoundness: Reports `Theorem` for an invalid F.
(Derives *false* for a satisfiable $\mathcal{S}$.)

Different Ways of Being Incorrect

unsoundness: Reports `Theorem` for an invalid F.
(Derives *false* for a satisfiable $\mathcal{S}$.)

- Check the proof and see what went wrong.

Different Ways of Being Incorrect

unsoundness: Reports `Theorem` for an invalid F.
(Derives *false* for a satisfiable $\mathcal{S}$.)

- Check the proof and see what went wrong.

completeness issue: Reports `Non-theorem` for a valid F.
(Finitely saturates unsat. $\mathcal{S}$ without deriving *false*.)

Different Ways of Being Incorrect

unsoundness: Reports `Theorem` for an invalid F.
(Derives *false* for a satisfiable $\mathcal{S}$.)

- Check the proof and see what went wrong.

completeness issue: Reports `Non-theorem` for a valid F.
(Finitely saturates unsat. $\mathcal{S}$ without deriving *false*.)

- Should have said `Unknown` here!

Different Ways of Being Incorrect

unsoundness: Reports `Theorem` for an invalid F.
(Derives *false* for a satisfiable $\mathcal{S}$.)

- Check the proof and see what went wrong.

completeness issue: Reports `Non-theorem` for a valid F.
(Finitely saturates unsat. $\mathcal{S}$ without deriving *false*.)

- Should have said `Unknown` here!

fairness issue: Prover runs indefinitely, while a proof exists.
(Violation of fairness criteria in saturation.)

Different Ways of Being Incorrect

unsoundness: Reports `Theorem` for an invalid F.
(Derives *false* for a satisfiable $\mathcal{S}$.)

- Check the proof and see what went wrong.

completeness issue: Reports `Non-theorem` for a valid F.
(Finitely saturates unsat. $\mathcal{S}$ without deriving *false*.)

- Should have said `Unknown` here!

fairness issue: Prover runs indefinitely, while a proof exists.
(Violation of fairness criteria in saturation.)

- never (strictly) violated after finitely many steps

Violating the Contract of Proper Behaviour

General error conditions shared by any other program:

program crash E.g.,

Violating the Contract of Proper Behaviour

General error conditions shared by any other program:

program crash E.g.,

- unhandled exceptions

Violating the Contract of Proper Behaviour

General error conditions shared by any other program:

program crash E.g.,

- unhandled exceptions
- signal interrupts (SIGFPE, SIGSEG)

Violating the Contract of Proper Behaviour

General error conditions shared by any other program:

program crash E.g.,

- unhandled exceptions
- signal interrupts (SIGFPE, SIGSEG)

assertion violation defensive development via assertions

- around 2500 assertions in total;
  (one per 77 lines on average)
- potential errors detected early on

Outline

Input Search Space

Problem input space:

- infinite, in principle
- in practice, we sample representative benchmarks, e.g.:
  - the TPTP library (∼20k problems)
  - SMT-LIB (∼46k relevant problems)

Input Search Space

Problem input space:

- infinite, in principle
- in practice, we sample representative benchmarks, e.g.:
  - the TPTP library (∼20k problems)
  - SMT-LIB (∼46k relevant problems)

Configuration space:

- around 75 proof search parameters
  (boolean, multi-valued, numeric)
- the search space $> 2^{75}$
  (too large to explore systematically)

The Debugging Process

Bug reports:

- from users / non-core developers; via email
- from random testing (dedicated cluster)

The Debugging Process

Bug reports:

- from users / non-core developers; via email
- from random testing (dedicated cluster)

Debugger's inventory:

the good old: `cout << "Here" << endl;`

The Debugging Process

Bug reports:

- from users / non-core developers; via email
- from random testing (dedicated cluster)

Debugger's inventory:

the good old: `cout << "Here" << endl;`

tracing home-made library of tracing macros

- `CALL` added at the start of each function
- explicit stack maintained

The Debugging Process

Bug reports:

- from users / non-core developers; via email
- from random testing (dedicated cluster)

Debugger's inventory:

the good old: `cout << "Here" << endl;`

tracing — home-made library of tracing macros

- `CALL` added at the start of each function
- explicit stack maintained

memory checking — own memory manager

- tune performance, enforce memory limits
- memory leak reporting

The Debugging Process

Bug reports:

- from users / non-core developers; via email
- from random testing (dedicated cluster)

Debugger's inventory:

the good old: `cout << "Here" << endl;`

tracing home-made library of tracing macros

- `CALL` added at the start of each function
- explicit stack maintained

memory checking own memory manager

- tune performance, enforce memory limits
- memory leak reporting

silent memory issues and segmentation faults

- one of the most difficult kinds to resolve
- Valgrind is usually of great help here

Proof Checking

Independent way of verifying `Theorem` results (checking soundness)

Proof Checking

Independent way of verifying `Theorem` results (checking soundness)

Example (Consider the following input:)

$$p(a) \qquad \neg p(x) \vee b = x \qquad \neg p(b)$$

Proof Checking

Independent way of verifying `Theorem` results (checking soundness)

Example (Consider the following input:)

$$p(a) \qquad \neg p(x) \vee b = x \qquad \neg p(b)$$

following proof in TPTP format produced

```
1. p(a) [input]
2. ~p(X0) | b = X0 [input]
3. ~p(b) [input]
4. a = b [resolution 2,1]
5. ~p(a) [backward demodulation 4,3]
7. $false [subsumption resolution 5,1]
```

Proof Checking

Independent way of verifying `Theorem` results (checking soundness)

Example (Consider the following input:)

$$p(a) \qquad \neg p(x) \vee b = x \qquad \neg p(b)$$

following proof in TPTP format produced

```
1. p(a) [input]
2. ~p(X0) | b = X0 [input]
3. ~p(b) [input]
4. a = b [resolution 2,1]
5. ~p(a) [backward demodulation 4,3]
7. $false [subsumption resolution 5,1]
```

"`vampire -p proofcheck`" for step 5:

```
fof(pr4,axiom, a = b ).
fof(pr3,axiom, ~p(b) ).
fof(r5,conjecture, ~p(a) ).
```

Outline

Full and Automated Proof Checking

Currently, proof checking skips:

Full and Automated Proof Checking

Currently, proof checking skips:

- symbol introducing preprocessing
  - e.g., Skolemization, formula naming, ...
  - does not preserve logical equivalence (only equisatisfiability)
  - global "freshness" condition

Full and Automated Proof Checking

Currently, proof checking skips:

- symbol introducing preprocessing
  - e.g., Skolemization, formula naming, ...
  - does not preserve logical equivalence (only equisatisfiability)
  - global "freshness" condition
- inferences backed by SAT and SMT solving
  - as of now, trusted as black boxes

Full and Automated Proof Checking

Currently, proof checking skips:

- symbol introducing preprocessing
  - e.g., Skolemization, formula naming, ...
  - does not preserve logical equivalence (only equisatisfiability)
  - global "freshness" condition
- inferences backed by SAT and SMT solving
  - as of now, trusted as black boxes

Level of detail provided:

- more details in the proof $\longrightarrow$ larger overhead

Full and Automated Proof Checking

Currently, proof checking skips:

- symbol introducing preprocessing
  - e.g., Skolemization, formula naming, ...
  - does not preserve logical equivalence (only equisatisfiability)
  - global "freshness" condition
- inferences backed by SAT and SMT solving
  - as of now, trusted as black boxes

Level of detail provided:

- more details in the proof $\longrightarrow$ larger overhead
- independent checking prover may fail to reprove a step

Full and Automated Proof Checking

Currently, proof checking skips:

- symbol introducing preprocessing
  - e.g., Skolemization, formula naming, ...
  - does not preserve logical equivalence (only equisatisfiability)
  - global "freshness" condition
- inferences backed by SAT and SMT solving
  - as of now, trusted as black boxes

Level of detail provided:

- more details in the proof $\longrightarrow$ larger overhead
- independent checking prover may fail to reprove a step

Ideally, ...

...strive for a standalone proof format with formal semantics!

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

- specific instantiation of the calculus

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

- specific instantiation of the calculus
- all skipped inferences / removed clauses must be redundant

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

- specific instantiation of the calculus
- all skipped inferences / removed clauses must be redundant
- saturated sets tend to be much larger than proofs!

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

- specific instantiation of the calculus
- all skipped inferences / removed clauses must be redundant
- saturated sets tend to be much larger than proofs!

➥ Currently, no standard way for certifying satisfiability!

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

- specific instantiation of the calculus
- all skipped inferences / removed clauses must be redundant
- saturated sets tend to be much larger than proofs!

➥ Currently, no standard way for certifying satisfiability!

A related challenge – monitoring fairness

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

- specific instantiation of the calculus
- all skipped inferences / removed clauses must be redundant
- saturated sets tend to be much larger than proofs!

➥ Currently, no standard way for certifying satisfiability!

A related challenge – monitoring fairness

- liveness property - strictly speaking impossible to monitor

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

- specific instantiation of the calculus
- all skipped inferences / removed clauses must be redundant
- saturated sets tend to be much larger than proofs!

➥ Currently, no standard way for certifying satisfiability!

A related challenge – monitoring fairness

- liveness property - strictly speaking impossible to monitor
- strengthen to bounded fairness, e.g.
  clause of age A will be processed no later than after kA steps

Handling Non-theorem Results

A challenging research topic

How to practically check that a saturated set is indeed saturated?

- specific instantiation of the calculus
- all skipped inferences / removed clauses must be redundant
- saturated sets tend to be much larger than proofs!

➥ Currently, no standard way for certifying satisfiability!

A related challenge – monitoring fairness

- liveness property - strictly speaking impossible to monitor
- strengthen to bounded fairness, e.g.
  clause of age A will be processed no later than after kA steps

➥ turned into a response property

Achieving Better Coverage with Random Testing

How to improve the current random sampling approach?

Achieving Better Coverage with Random Testing

How to improve the current random sampling approach?

parameter space coverage

- direct random sampling to under-tested areas?
- target new features / untested combinations

Achieving Better Coverage with Random Testing

How to improve the current random sampling approach?

parameter space coverage

- direct random sampling to under-tested areas?
- target new features / untested combinations

problem space coverage

- little work done so far
- libraries may lack inputs for testing a new feature
- experiment with fuzzing?

Conclusion

Summary:

- described challenges in testing an automated theorem prover
- based on experience with Vampire
- generalises to other ATPs

Conclusion

Summary:

- described challenges in testing an automated theorem prover
- based on experience with Vampire
- generalises to other ATPs

Concrete future work:

- 100% reliable proof checking
- better input problem coverage

Conclusion

Summary:

- described challenges in testing an automated theorem prover
- based on experience with Vampire
- generalises to other ATPs

Concrete future work:

- 100% reliable proof checking
- better input problem coverage

Thank you for your attention!