Instantiation-Based Automated Theorem Proving for First-Order Logic

Konstantin Korovin

The University of Manchester
UK

korovin@cs.man.ac.uk

Theorem proving for first-order logic

Theorem proving: Show that a given first-order formula is a theorem.

Maths: Axioms of groups *Group*

- $\forall x, y, z \ \ (x \cdot (y \cdot z) \simeq (x \cdot y) \cdot z)$
- $\forall x \ \ (x \cdot x^{-1} \simeq e)$
- $\forall x \ \ (x \cdot e \simeq x)$

Consider $F = \forall x \exists y \ ((x \cdot y)^{-1} \simeq y^{-1} \cdot x^{-1})$

Is F a theorem in the group theory: $\mathit{Group} \models F$?

Theorem proving for first-order logic

Theorem proving: Show that a given first-order formula is a theorem.

Maths: Axioms of groups *Group*

- $\forall x, y, z \ \ (x \cdot (y \cdot z) \simeq (x \cdot y) \cdot z)$
- $\forall x \ \ (x \cdot x^{-1} \simeq e)$
- $\forall x \ \ (x \cdot e \simeq x)$

Consider $F = \forall x \exists y \ ((x \cdot y)^{-1} \simeq y^{-1} \cdot x^{-1})$

Is F a theorem in the group theory: $Group \models F$?

Verification: Axioms of arrays

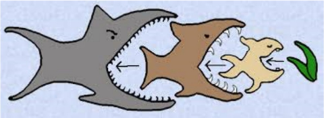
- $\forall a, i, e \ (select(store(a, i, e), i) \simeq e)$
- $\forall a, i, j, e \ (i \not\simeq j \rightarrow (select(store(a, i, e), j) \simeq select(a, j)))$
- $\forall a_1, a_2 \ ((\forall i \ (select(a_1, i) \simeq select(a_2, i))) \rightarrow a_1 \simeq a_2)$

Is $\exists a \exists i \forall j \ (select(a, i) \simeq select(a, j))$ a theorem in the theory of arrays ?

Why first-order logic

- Expressive most of mathematics can be formalised in FOL
- Complete calculi – uniform reasoning methods
- Efficient reasoning – well-understood algorithms and datastructures
- Reductions from HOL to FOL: Blanchette (Sledgehammer), Urban (Mizar)

FOL provides a good balance between expressivity and efficiency.

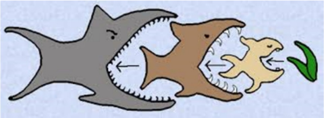


*

Why first-order logic

- Expressive most of mathematics can be formalised in FOL
- Complete calculi – uniform reasoning methods
- Efficient reasoning – well-understood algorithms and datastructures
- Reductions from HOL to FOL: Blanchette (Sledgehammer), Urban (Mizar)

FOL provides a good balance between expressivity and efficiency.

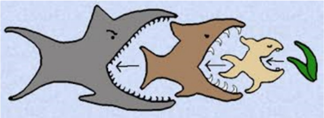
 *

["The Unreasonable Effectiveness of Mathematics in the Natural Sciences" E. Wigner]

Why first-order logic

- Expressive most of mathematics can be formalised in FOL
- Complete calculi – uniform reasoning methods
- Efficient reasoning – well-understood algorithms and datastructures
- Reductions from HOL to FOL: Blanchette (Sledgehammer), Urban (Mizar)

FOL provides a good balance between expressivity and efficiency.


*

["The Unreasonable Effectiveness of Mathematics in the Natural Sciences" E. Wigner]

[Unreasonable effectiveness of logic in computer science]

Calculi for first-order logic

Calculi complete for first-order logic:

- natural deduction
  - difficult to automate
- tableaux-based calculi
  - popular with special fragments: modal and description logics
  - difficult to automate efficiently in the general case
- resolution/superposition calculi
  - general purpose
  - can be efficiently automated
  - decision procedure for many fragments
- instantiation-based calculi
  - combination of efficient propositional reasoning with first-order reasoning
  - can be efficiently automated
  - decision procedure for the effectively propositional fragment (EPR)

Calculi for first-order logic

Calculi complete for first-order logic:

- natural deduction
  - difficult to automate
- tableaux-based calculi
  - popular with special fragments: modal and description logics
  - difficult to automate efficiently in the general case
- resolution/superposition calculi
  - general purpose
  - can be efficiently automated
  - decision procedure for many fragments
- instantiation-based calculi
  - combination of efficient propositional reasoning with first-order reasoning
  - can be efficiently automated
  - decision procedure for the effectively propositional fragment (EPR)

Calculi for first-order logic

Calculi complete for first-order logic:

- natural deduction
  - difficult to automate
- tableaux-based calculi
  - popular with special fragments: modal and description logics
  - difficult to automate efficiently in the general case
- resolution/superposition calculi
  - general purpose
  - can be efficiently automated
  - decision procedure for many fragments
- instantiation-based calculi
  - combination of efficient propositional reasoning with first-order reasoning
  - can be efficiently automated
  - decision procedure for the effectively propositional fragment (EPR)

Calculi for first-order logic

Calculi complete for first-order logic:

- natural deduction
  - difficult to automate
- tableaux-based calculi
  - popular with special fragments: modal and description logics
  - difficult to automate efficiently in the general case
- resolution/superposition calculi
  - general purpose
  - can be efficiently automated
  - decision procedure for many fragments
- instantiation-based calculi
  - combination of efficient propositional reasoning with first-order reasoning
  - can be efficiently automated
  - decision procedure for the effectively propositional fragment (EPR)

Refutational theorem proving

Theorem proving:

$$\models \text{Axioms} \rightarrow \text{Theorem}$$

Refutational theorem proving:

$$\text{Axioms} \wedge \neg\text{Theorem} \models \bot$$

Other reasoning problems: validity, equivalence etc can be reduced to (un)satisfiability

In order to apply efficient reasoning methods we need to transform formulas into equi-satisfiable conjunctive normal form.

CNF transformation

Main steps in the basic CNF transformation:

1. Prenex normal form – moving all quantifiers up-front
   $\forall y\ [\forall x\ [p(f(x), y)] \rightarrow \forall v \exists z\ [q(f(z)) \wedge p(v, z)]] \Rightarrow$
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))]$

CNF transformation

Main steps in the basic CNF transformation:

1. Prenex normal form – moving all quantifiers up-front
   $\forall y\ [\forall x\ [p(f(x), y)] \rightarrow \forall v \exists z\ [q(f(z)) \wedge p(v, z)]] \Rightarrow$
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))]$
2. Skolemization – eliminating existential quantifiers
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))] \Rightarrow$
   $\forall y \forall v\ [p(f(sk_1(y)), y) \rightarrow (q(f(sk_2(y, v))) \wedge p(v, sk_2(y, v)))]$

CNF transformation

Main steps in the basic CNF transformation:

1. Prenex normal form – moving all quantifiers up-front
   $\forall y\ [\forall x\ [p(f(x), y)] \rightarrow \forall v \exists z\ [q(f(z)) \wedge p(v, z)]] \Rightarrow$
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))]$
2. Skolemization – eliminating existential quantifiers
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))] \Rightarrow$
   $\forall y \forall v\ [p(f(sk_1(y)), y) \rightarrow (q(f(sk_2(y, v))) \wedge p(v, sk_2(y, v)))]$
3. CNF transformation of the quantifier-free part
   $\forall y \forall v\ [\quad p(f(sk_1(y)), y) \rightarrow (q(f(sk_2(y, v))) \wedge p(v, sk_2(y, v)))] \Rightarrow$
   $\forall y \forall v\ [\quad (\neg p(f(sk_1(y)), y) \vee q(f(sk_2(y, v)))) \wedge$
   $\qquad (\neg p(f(sk_1(y)), y) \vee p(v, sk_2(y, v)))]$

CNF transformation

Main steps in the basic CNF transformation:

1. Prenex normal form – moving all quantifiers up-front
   $\forall y\ [\forall x\ [p(f(x), y)] \rightarrow \forall v \exists z\ [q(f(z)) \wedge p(v, z)]] \Rightarrow$
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))]$
2. Skolemization – eliminating existential quantifiers
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))] \Rightarrow$
   $\forall y \forall v\ [p(f(sk_1(y)), y) \rightarrow (q(f(sk_2(y, v))) \wedge p(v, sk_2(y, v)))]$
3. CNF transformation of the quantifier-free part
   $\forall y \forall v\ [\quad p(f(sk_1(y)), y) \rightarrow (q(f(sk_2(y, v))) \wedge p(v, sk_2(y, v)))] \Rightarrow$
   $\forall y \forall v\ [\quad (\neg p(f(sk_1(y)), y) \vee q(f(sk_2(y, v)))) \wedge$
   $\qquad (\neg p(f(sk_1(y)), y) \vee p(v, sk_2(y, v)))]$

CNF transformation

Main steps in the basic CNF transformation:

1. Prenex normal form – moving all quantifiers up-front
   $\forall y\ [\forall x\ [p(f(x), y)] \rightarrow \forall v \exists z\ [q(f(z)) \wedge p(v, z)]] \Rightarrow$
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))]$
2. Skolemization – eliminating existential quantifiers
   $\forall y \exists x \forall v \exists z\ [p(f(x), y) \rightarrow (q(f(z)) \wedge p(v, z))] \Rightarrow$
   $\forall y \forall v\ [p(f(sk_1(y)), y) \rightarrow (q(f(sk_2(y, v))) \wedge p(v, sk_2(y, v)))]$
3. CNF transformation of the quantifier-free part
   $\forall y \forall v\ [\quad p(f(sk_1(y)), y) \rightarrow (q(f(sk_2(y, v))) \wedge p(v, sk_2(y, v)))] \Rightarrow$
   $\forall y \forall v\ [\quad (\neg p(f(sk_1(y)), y) \vee q(f(sk_2(y, v)))) \wedge$
   $\qquad (\neg p(f(sk_1(y)), y) \vee p(v, sk_2(y, v)))]$

Main reasoning problem:

Given set of clauses S prove that it (un)satisfiable.

Inference systems: propositional resolution

Inference-based theorem proving

Given: S – set of clauses.

Example: $S = \{q \vee \neg p, \quad p \vee q, \quad \neg q\}$

We want to prove that S is unsatisfiable.

Inference-based theorem proving

Given: S – set of clauses.

Example: $S = \{q \vee \neg p, \quad p \vee q, \quad \neg q\}$

We want to prove that S is unsatisfiable.

Inference-based theorem proving

Given: S – set of clauses.

Example: $S = \{q \vee \neg p, \quad p \vee q, \quad \neg q\}$

We want to prove that S is unsatisfiable.

General Idea:

- use a set of simple rules for deriving new logical consequences from S.
- use these inference rules to derive the contradiction signified by the empty clause $\Box$

Propositional Resolution

Propositional Resolution inference system $\mathbb{BR}$, consists of the following inference rules:

- Binary Resolution Rule (BR):

$$\frac{C \vee p \qquad \neg p \vee D}{C \vee D}\,(BR)$$

- Binary Factoring Rule (BF):

$$\frac{C \vee L \vee L}{C \vee L}\,(BF)$$

where L is a literal.

Example

Given: $S = \{q \vee \neg p, \quad p \vee q, \quad \neg q\}$

A proof in resolution calculus:

$$\dfrac{\dfrac{\dfrac{q \vee \neg p \qquad p \vee q}{q \vee q}\,\text{(BR)}}{q}\,\text{(BF)} \qquad \neg q}{\Box}\,\text{(BR)}$$

Soundness/Completeness

Theorem (Soundness)

Resolution is a sound inference system:

$$S \vdash_{BR} \Box \quad \textit{implies} \quad S \models \bot$$

Soundness/Completeness

Theorem (Soundness)

Resolution is a sound inference system:

$$S \vdash_{BR} \Box \quad \textit{implies} \quad S \models \bot$$

Theorem (Completeness)

Resolution is a complete inference system:

$$S \models \bot \quad \textit{implies} \quad S \vdash_{BR} \Box$$

Proof search based on inference systems

Basic approach. A Saturation Process:

Given set of clauses S we exhaustively apply all inference rules adding the conclusions to this set until the contradiction ($\square$) is derived.

$$S_0 \Rightarrow S_1 \Rightarrow \ldots S_n \Rightarrow \ldots$$

Proof search based on inference systems

Basic approach. A Saturation Process:

Given set of clauses S we exhaustively apply all inference rules adding the conclusions to this set until the contradiction ($\Box$) is derived.

$$S_0 \Rightarrow S_1 \Rightarrow \ldots S_n \Rightarrow \ldots$$

Three outcomes:

1. $\Box$ is derived ($\Box \in S_n$ for some n), then S is unsatisfiable (soundness);
2. no new clauses can be derived from S and $\perp \notin S$, then S is saturated; in this case S is satisfiable, (completeness).
3. S grows ad infinitum, the process does not terminate.

Proof search based on inference systems

Basic approach. A Saturation Process:

Given set of clauses S we exhaustively apply all inference rules adding the conclusions to this set until the contradiction ($\square$) is derived.

$$S_0 \Rightarrow S_1 \Rightarrow \dots S_n \Rightarrow \dots$$

Three outcomes:

1. $\square$ is derived ($\square \in S_n$ for some n), then S is unsatisfiable (soundness);
2. no new clauses can be derived from S and $\perp \notin S$, then S is saturated; in this case S is satisfiable, (completeness).
3. S grows ad infinitum, the process does not terminate.

The main challenge: speed up the first two cases and reduce non-termination.

First-order resolution

Herbrand theorem

First-order clauses S:

$$p(a) \vee q(a, f(b))$$
$$\forall x, y \; [\neg p(x) \vee \neg q(x, f(y))]$$
$$\ldots$$

How to check if S is (un)satisfiable ?

Herbrand theorem

First-order clauses S:

$$p(a) \vee q(a, f(b))$$
$$\forall x, y \; [\neg p(x) \vee \neg q(x, f(y))]$$
$$\ldots$$

How to check if S is (un)satisfiable ?

Theorem (Herbrand)

S is unsatisfiable if and only there is a finite set of ground instances of clauses in S which are propositionally unsatisfiable.

Herbrand theorem

First-order clauses S:

$$p(a) \vee q(a, f(b))$$
$$\forall x, y\ [\neg p(x) \vee \neg q(x, f(y))]$$
$$\dots$$

How to check if S is (un)satisfiable ?

Theorem (Herbrand)

S is unsatisfiable if and only there is a finite set of ground instances of clauses in S which are propositionally unsatisfiable.

General approach: enumerate ground instances and apply resolution to the ground instances.

Herbrand theorem

First-order clauses S:

$$p(a) \vee q(a, f(b))$$
$$\neg p(z)$$
$$\neg q(x, f(y))$$

How to check if S is (un)satisfiable ?

Replace variables by ground terms and apply resolution:

$$\neg q(a, f(a))$$
$$\neg q(b, f(f(a)))$$
$$\dots$$
$$\neg q(a, f(b))$$

Herbrand theorem

First-order clauses S:

$$p(a) \vee q(a, f(b))$$
$$\neg p(z)$$
$$\neg q(x, f(y))$$

How to check if S is (un)satisfiable ?

Replace variables by ground terms and apply resolution:

$$\neg q(a, f(a))$$
$$\neg q(b, f(f(a)))$$
$$\ldots$$
$$\neg q(a, f(b))$$

Herbrand theorem

First-order clauses S:

$$p(a) \vee q(a, f(b))$$
$$\neg p(z)$$
$$\neg q(x, f(y))$$

How to check if S is (un)satisfiable ?

Replace variables by ground terms and apply resolution:

$$\neg q(a, f(a))$$
$$\neg q(b, f(f(a)))$$
$$\cdots$$
$$\neg q(a, f(b))$$
$$p(a) \qquad (BR)$$

Herbrand theorem

First-order clauses S:

$$p(a) \vee q(a, f(b))$$
$$\neg p(z)$$
$$\neg q(x, f(y))$$

How to check if S is (un)satisfiable ?

Replace variables by ground terms and apply resolution:

$$\neg q(a, f(a))$$
$$\neg q(b, f(f(a)))$$
$$\dots$$
$$\neg q(a, f(b))$$
$$p(a) \qquad (BR)$$
$$\neg p(a)$$

Herbrand theorem

First-order clauses S:

$$p(a) \vee q(a, f(b))$$

$$\neg p(z)$$

$$\neg q(x, f(y))$$

How to check if S is (un)satisfiable ?

Replace variables by ground terms and apply resolution:

$$\neg q(a, f(a))$$

$$\neg q(b, f(f(a)))$$

$$\dots$$

$$\neg q(a, f(b))$$

$$p(a) \qquad (BR)$$

$$\neg p(a)$$

$$\square \qquad (BR)$$

Non-ground resolution

- A non-ground clause can be seen as representation of a (possibly infinite) set of its ground instances.
- Consider $q(x, a) \vee \underline{p(x)}$ and $q(y, z) \vee \neg\underline{p(f(y))}$.

Non-ground resolution

- A non-ground clause can be seen as representation of a (possibly infinite) set of its ground instances.
- Consider $q(x, a) \vee \underline{p(x)}$ and $q(y, z) \vee \neg\underline{p(f(y))}$.
  A common instance to which ground resolution is applicable:
  $q(f(a), a) \vee \underline{p(f(a))}$ and $q(a, a) \vee \neg\underline{p(f(a))}$

Non-ground resolution

- A non-ground clause can be seen as representation of a (possibly infinite) set of its ground instances.
- Consider $q(x, a) \vee \underline{p(x)}$ and $q(y, z) \vee \neg\underline{p(f(y))}$.
  A common instance to which ground resolution is applicable:
  $q(f(a), a) \vee \underline{p(f(a))}$ and $q(a, a) \vee \neg\underline{p(f(a))}$
- There are other ground instances e.g.:
  $q(f(f(a)), a) \vee \underline{p(f(f(a)))}$ and $q(f(a), f(f(f(a))) \vee \neg\underline{p(f(f(a))}$

Non-ground resolution

- A non-ground clause can be seen as representation of a (possibly infinite) set of its ground instances.
- Consider $q(x, a) \vee \underline{p(x)}$ and $q(y, z) \vee \neg\underline{p(f(y))}$.
  A common instance to which ground resolution is applicable:
  $q(f(a), a) \vee \underline{p(f(a))}$ and $q(a, a) \vee \neg\underline{p(f(a))}$
- There are other ground instances e.g.:
  $q(f(f(a)), a) \vee \underline{p(f(f(a)))}$ and $q(f(a), f(f(f(a))) \vee \neg\underline{p(f(f(a))}$
- In order to apply ground resolution we need find substitution which make atoms $\underline{p(x)}$ and $\underline{p(f(y))}$ syntactically equal.

Non-ground resolution

- A non-ground clause can be seen as representation of a (possibly infinite) set of its ground instances.
- Consider $q(x,a) \vee \underline{p(x)}$ and $q(y,z) \vee \neg\underline{p(f(y))}$.
  A common instance to which ground resolution is applicable:
  $q(f(a),a) \vee \underline{p(f(a))}$ and $q(a,a) \vee \neg\underline{p(f(a))}$
- There are other ground instances e.g.:
  $q(f(f(a)),a) \vee \underline{p(f(f(a)))}$ and $q(f(a),f(f(f(a))) \vee \neg\underline{p(f(f(a))}$
- In order to apply ground resolution we need find substitution which make atoms $\underline{p(x)}$ and $\underline{p(f(y))}$ syntactically equal.
- Such substitutions are called unifiers.

Non-ground resolution

- A non-ground clause can be seen as representation of a (possibly infinite) set of its ground instances.
- Consider $q(x, a) \vee \underline{p(x)}$ and $q(y, z) \vee \neg\underline{p(f(y))}$.
  A common instance to which ground resolution is applicable:
  $q(f(a), a) \vee \underline{p(f(a))}$ and $q(a, a) \vee \neg\underline{p(f(a))}$
- There are other ground instances e.g.:
  $q(f(f(a)), a) \vee \underline{p(f(f(a)))}$ and $q(f(a), f(f(f(a))) \vee \neg\underline{p(f(f(a))}$
- In order to apply ground resolution we need find substitution which make atoms $\underline{p(x)}$ and $\underline{p(f(y))}$ syntactically equal.
- Such substitutions are called unifiers.
- Even for two clauses there are infinite number of possible instances to which resolution is applicable.

Most general unifiers

- Consider $q(x, a) \vee \underline{p(x)}$ and $q(y, z) \vee \neg\underline{p(f(y))}$
- substitute $\sigma = \{x \mapsto f(y)\}$
- then $q(f(y), a) \vee \underline{p(f(y))}$ and $q(y, z) \vee \neg\underline{p(f(y))}$.
- Note:
    1. underlined atoms are syntactically equal
    2. any other substitution can be seen as an instance of σ

       σ – most general unifier $\sigma = \mathrm{mgu}(p(x), p(f(y)))$
    3. σ can be seen as a finite representation of all infinitely many substitutions which makes terms equal.

Most general unifiers

- Consider $q(x,a) \vee \underline{p(x)}$ and $q(y,z) \vee \neg\underline{p(f(y))}$
- substitute $\sigma = \{x \mapsto f(y)\}$
- then $q(f(y),a) \vee \underline{p(f(y))}$ and $q(y,z) \vee \neg\underline{p(f(y))}$.
- Note:
  1. underlined atoms are syntactically equal
  2. any other substitution can be seen as an instance of σ

     σ – most general unifier $\sigma = \mathrm{mgu}(p(x), p(f(y)))$
  3. σ can be seen as a finite representation of all infinitely many substitutions which makes terms equal.

Theorem [Robinson 1965] If two atoms $p(t(\bar{x}))$ and $p(s(\bar{x}))$ have a common ground instance then there is a unique most general unifier σ, which can be effectively computed. Note $p(t(\bar{x}))\sigma = p(s(\bar{x}))\sigma$.

First-order resolution:

- Resolution rule (BR):

$$\frac{C \vee p \qquad \neg p' \vee D}{(C \vee D)\sigma}\ (BR)$$

  where $\sigma = \mathrm{mgu}(p, p')$

- Example:

$$\frac{q(x,a) \vee \underline{p(x)} \qquad q(y,z) \vee \neg \underline{p(f(y))}}{q(f(y),a) \vee q(y,z)}\ (BR)$$

  where $\mathrm{mgu}(p(x), p(f(y))) = \{x \mapsto f(y)\}$

First-order resolution:

- Resolution rule (BR):

$$\frac{C \vee p \qquad \neg p' \vee D}{(C \vee D)\sigma}\ (BR)$$

where $\sigma = \mathrm{mgu}(p, p')$

- Example:

$$\frac{q(x,a) \vee \underline{p(x)} \qquad q(y,z) \vee \neg \underline{p(f(y))}}{q(f(y),a) \vee q(y,z)}\ (BR)$$

where $\mathrm{mgu}(p(x), p(f(y))) = \{x \mapsto f(y)\}$

Theorem [Bachmair, Ganzinger] Resolution with many refinements is complete for first-order logic.

The magic of resolution

Resolution calculus with appropriate simplifications, selection functions and saturation strategies is a decision procedure for many fragments:

- monadic fragment [Bachmair, Ganzinger, Waldmann]
- modal logic translations [Hustadt, Schmidt]
- guarded fragment [Ganzinger, de Nivelle]
- two variable fragment [de Nivelle, Pratt-Hartmann]
- fluted fragment [Hustadt, Schmidt, Georgieva]
- many description logic fragments [Kazakov, Motik, Sattler, ...]
- ...

The magic of resolution

Resolution calculus with appropriate simplifications, selection functions and saturation strategies is a decision procedure for many fragments:

- monadic fragment [Bachmair, Ganzinger, Waldmann]
- modal logic translations [Hustadt, Schmidt]
- guarded fragment [Ganzinger, de Nivelle]
- two variable fragment [de Nivelle, Pratt-Hartmann]
- fluted fragment [Hustadt, Schmidt, Georgieva]
- many description logic fragments [Kazakov, Motik, Sattler, ...]
- ...

- Original proofs of decidability for these fragments are based on diverse, complicated, model theoretic arguments.
- Resolution-based methods provide practical procedures
- Vampire, E, SPASS are based on extensions resolution

Modular instantiation-based reasoning

SAT/SMT vs First-Order

The main reasoning problem:

Check that a given a set of clauses S is (un)satisfiable.

Ground (SAT/SMT)

$$bv(a) \vee mem(c, d)$$
$$\neg bv(a) \vee mem(d, c)$$

Very efficient solvers

Not very expressive

CDCL/Congruence closure

First-Order

$$\forall x \exists y \quad \neg mem_1(x, y) \vee mem_2(y, f(x))$$
$$bv(a) \vee mem(d, c)$$

Very expressive

Ground: not as efficient

Resolution/Superposition

From ground to first-order: Efficient at ground + Expressive?

Resolution weaknesses

Resolution :

$$\frac{C \vee L \qquad \overline{L'} \vee D}{(C \vee D)\sigma}$$

Example :

$$\frac{Q(x) \vee P(x) \qquad \neg P(a) \vee R(y)}{Q(a) \vee R(y)}$$

$$\boxed{\begin{array}{c} L_1 \vee C_1 \\ \vdots \\ L_n \vee C_n \end{array}}$$

Weaknesses:

- Inefficient in propositional case
- Proof search without model search
- Length of clauses can grow fast
- Recombination of clauses
- No effective model representation

Basic idea behind instantiation proving

Can we approximate first-order by ground reasoning?

Basic idea behind instantiation proving

Can we approximate first-order by ground reasoning?

Theorem (Herbrand). S is unsatisfiable if and only there is a finite set of ground instances of clauses of S which are propositionally unsatisfiable.

Basic idea: Interleave instantiation with propositional reasoning.

Main issues:

- How to restrict instantiations.
- How to interleave instantiation with propositional reasoning.

Basic idea behind instantiation proving

Can we approximate first-order by ground reasoning?

Theorem (Herbrand). S is unsatisfiable if and only there is a finite set of ground instances of clauses of S which are propositionally unsatisfiable.

Basic idea: Interleave instantiation with propositional reasoning.

Main issues:

- How to restrict instantiations.
- How to interleave instantiation with propositional reasoning.

[Wang'59; Gilmore'60; Plaisted'92; Inst-Gen Ganzinger, Korovin; Model Evolution Baumgartner Tinelli; AVATAR Voronkov; SGGS Bonacina Plaisted; Weidenbach,. . . , SMT quantifier instantiations Ge, de Moura, Reynolds. . .]

Overview of the Inst-Gen procedure

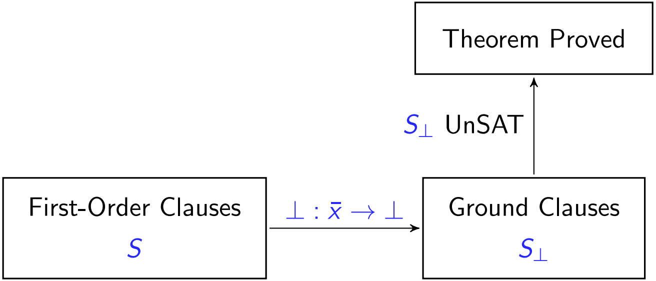
First-Order Clauses

S

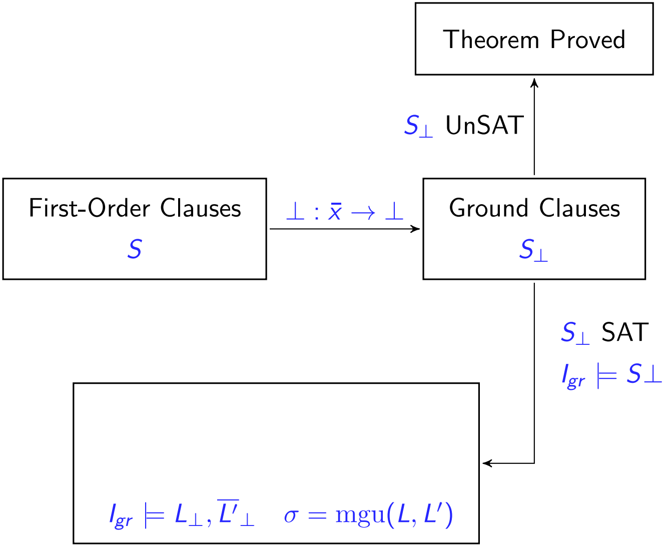
Overview of the Inst-Gen procedure

$$\boxed{\begin{array}{c}\text{First-Order Clauses}\\ S\end{array}} \xrightarrow{\perp \,:\, \bar{x} \to \perp} \boxed{\begin{array}{c}\text{Ground Clauses}\\ S_\perp\end{array}}$$

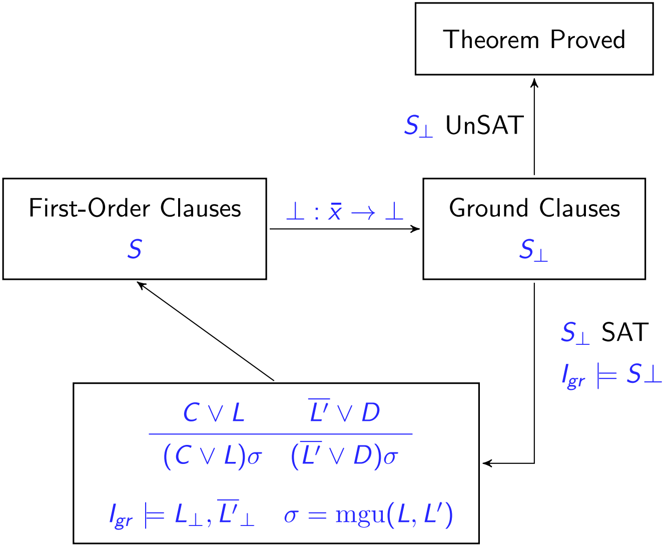
Overview of the Inst-Gen procedure



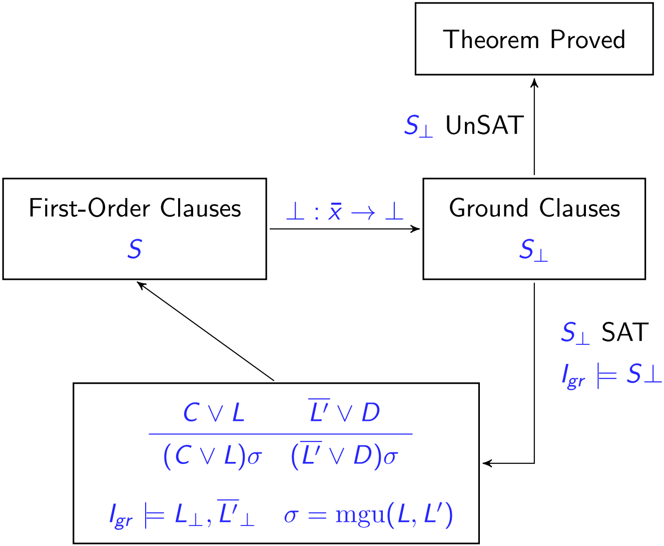
Overview of the Inst-Gen procedure



Overview of the Inst-Gen procedure



Overview of the Inst-Gen procedure



Theorem.(Ganzinger, Korovin) Inst-Gen is sound and complete for FOL.

Example:

$$p(f(x), b) \lor q(x, y)$$

$$\neg p(f(f(x)), y)$$

$$\neg q(f(x), x)$$

Example:

$$p(f(x), b) \vee q(x, y)$$

$$\neg p(f(f(x)), y)$$

$$\neg q(f(x), x)$$

$$p(f(\bot), b) \vee q(\bot, \bot)$$

$$\neg p(f(f(\bot)), \bot)$$

$$\neg q(f(\bot), \bot)$$

Example:

$$\boxed{\begin{array}{c} p(f(x), b) \vee q(x, y) \\ \neg p(f(f(x)), y) \\ \neg q(f(x), x) \end{array}}$$

$$\boxed{\begin{array}{c} p(f(\perp), b) \vee q(\perp, \perp) \\ \neg p(f(f(\perp)), \perp) \\ \neg q(f(\perp), \perp) \end{array}}$$

Example:

$$\boxed{\begin{array}{c} p(f(x), b) \vee q(x, y) \\ \neg p(f(f(x)), y) \\ \neg q(f(x), x) \end{array}}$$

$$\boxed{\begin{array}{c} p(f(\bot), b) \vee q(\bot, \bot) \\ \neg p(f(f(\bot)), \bot) \\ \neg q(f(\bot), \bot) \end{array}}$$

$$\boxed{\begin{array}{c} p(f(f(x)), b) \vee q(f(x), y) \\ \neg p(f(f(x)), b) \\ p(f(x), b) \vee q(x, y) \\ \neg p(f(f(x)), y) \\ \neg q(f(x), x) \end{array}}$$

Example:

$p(f(x), b) \vee q(x, y)$ $\neg p(f(f(x)), y)$ $\neg q(f(x), x)$	$p(f(\bot), b) \vee q(\bot, \bot)$ $\neg p(f(f(\bot)), \bot)$ $\neg q(f(\bot), \bot)$
$p(f(f(x)), b) \vee q(f(x), y)$ $\neg p(f(f(x)), b)$ $p(f(x), b) \vee q(x, y)$ $\neg p(f(f(x)), y)$ $\neg q(f(x), x)$	$p(f(f(\bot)), b) \vee q(f(\bot), \bot)$ $\neg p(f(f(\bot)), b)$ $p(f(\bot), b) \vee q(\bot, \bot)$ $\neg p(f(f(\bot)), \bot)$ $\neg q(f(\bot), \bot)$

Example:

$$\boxed{\begin{array}{c} p(f(x), b) \vee q(x, y) \\ \neg p(f(f(x)), y) \\ \neg q(f(x), x) \end{array}} \qquad \boxed{\begin{array}{c} p(f(\perp), b) \vee q(\perp, \perp) \\ \neg p(f(f(\perp)), \perp) \\ \neg q(f(\perp), \perp) \end{array}}$$

$$\boxed{\begin{array}{c} p(f(f(x)), b) \vee q(f(x), y) \\ \neg p(f(f(x)), b) \\ p(f(x), b) \vee q(x, y) \\ \neg p(f(f(x)), y) \\ \neg q(f(x), x) \end{array}} \qquad \boxed{\begin{array}{c} p(f(f(\perp)), b) \vee q(f(\perp), \perp) \\ \neg p(f(f(\perp)), b) \\ p(f(\perp), b) \vee q(\perp, \perp) \\ \neg p(f(f(\perp)), \perp) \\ \neg q(f(\perp), \perp) \end{array}}$$

The final set is propositionally unsatisfiable.

Resolution vs Inst-Gen

Resolution :

$$\frac{(C \vee L) \qquad (\overline{L'} \vee D)}{(C \vee D)\sigma}$$
$$\sigma = \mathrm{mgu}(L, L')$$

Weaknesses of resolution:

Proof search without model search

Inefficient in the ground/EPR case

Length of clauses can grow fast

Recombination of clauses

No explicit model representation

Instantiation :

$$\frac{(C \vee L) \qquad (\overline{L'} \vee D)}{(C \vee L)\sigma \qquad (\overline{L'} \vee D)\sigma}$$
$$\sigma = \mathrm{mgu}(L, L')$$

Strengths of instantiation:

Proof search guided by prop. models

Modular ground reasoning

Length of clauses is fixed

Decision procedure for EPR

No recombination

Redundancy elimination

Effective model representation

Redundancy Elimination (Inst-Gen)

The key to efficiency is redundancy elimination.

- usual: tautology elimination, strict subsumption
- global subsumption: non-ground simplifications using SAT/SMT reasoning
- blocking non-proper instantiators
- dismatching constraints
- predicate elimination
- sort inference/redundancies
- definitional redundancies
- ...

Redundancy Elimination

The key to efficiency is redundancy elimination.

Redundancy Elimination

The key to efficiency is redundancy elimination.

Ground clause C is redundant if

- $C_1, \ldots, C_n \models C$
- $C_1, \ldots, C_n \prec C$

Where $\prec$ is a well-founded ordering.

- $P(a) \models Q(b) \vee P(a)$
- $P(a) \prec \cancel{Q(b) \vee P(a)}$

Redundancy Elimination

The key to efficiency is redundancy elimination.

Ground clause C is redundant if

- $C_1, \ldots, C_n \models C$
- $C_1, \ldots, C_n \prec C$

- $P(a) \models Q(b) \vee P(a)$
- $P(a) \prec \cancel{Q(b) \vee P(a)}$

Where $\prec$ is a well-founded ordering.

Theorem Redundant clauses/closures can be eliminated.

Consequences:

- many usual redundancy elimination techniques
- redundancy for inferences
- new instantiation-specific redundancies

Simplifications by SAT/SMT solver *(K. IJCAR'08)*

Can off-the-shelf ground solver be used to simplify ground clauses?

Simplifications by SAT/SMT solver *(K. IJCAR'08)*

Can off-the-shelf ground solver be used to simplify ground clauses?

Abstract redundancy:

$C_1, \ldots, C_n \models C$ $\quad$ $S_{gr} \models C$ — ground solver

$C_1, \ldots, C_n \prec C$ $\quad$ follows from smaller ?

Simplifications by SAT/SMT solver *(K. IJCAR'08)*

Can off-the-shelf ground solver be used to simplify ground clauses?

Abstract redundancy:

$C_1, \ldots, C_n \models C$ — $S_{gr} \models C$ — ground solver

$C_1, \ldots, C_n \prec C$ — follows from smaller ?

Basic idea:

- split $D \subset C$
- check $S_{gr} \models D$
- add D to S and remove C

Simplifications by SAT/SMT solver *(K. IJCAR'08)*

Can off-the-shelf ground solver be used to simplify ground clauses?

Abstract redundancy:

$C_1, \ldots, C_n \models C$

$C_1, \ldots, C_n \prec C$

$S_{gr} \models C$ — ground solver

follows from smaller ?

Basic idea:

- split $D \subset C$
- check $S_{gr} \models D$
- add D to S and remove C

Global ground subsumption:

$$\frac{\cancel{D \vee C'}}{D}$$

where $S_{gr} \models D$ and $C' \neq \emptyset$

Global Ground Subsumption

S_{gr}	C
$\neg Q(a,b) \vee P(a) \vee P(b)$	$P(a) \vee Q(c,d) \vee Q(a,c)$
$P(a) \vee Q(a,b)$	
$\neg P(b)$	

Global Ground Subsumption

S_{gr}	C
$\neg Q(a,b) \lor P(a) \lor P(b)$	$P(a) \lor Q(c,d) \lor \cancel{Q(a,c)}$
$P(a) \lor Q(a,b)$	
$\neg P(b)$	

Global Ground Subsumption

S_{gr}	C
$\neg Q(a, b) \vee P(a) \vee P(b)$	$P(a) \vee \cancel{Q(c, d)} \vee \cancel{Q(a, c)}$
$P(a) \vee Q(a, b)$	
$\neg P(b)$	

A minimal $D \subset C$ such that $S_{gr} \models D$ can be found in a linear number of implication checks.

Global Ground Subsumption

S_{gr}	C
$\neg Q(a,b) \vee P(a) \vee P(b)$	$P(a) \vee \cancel{Q(c,d)} \vee \cancel{Q(a,c)}$
$P(a) \vee Q(a,b)$	
$\neg P(b)$	

A minimal $D \subset C$ such that $S_{gr} \models D$ can be found in a linear number of implication checks.

Global Ground Subsumption generalises:

- strict subsumption
- subsumption resolution
- ...

Non-ground simplifications by SAT/SMT *(K. IJCAR'08)*

Off-the-shelf SAT solver can be used to simplify ground clauses.

Can we also use SAT solver to simplify non-ground clauses?

Non-ground simplifications by SAT/SMT *(K. IJCAR'08)*

Off-the-shelf SAT solver can be used to simplify ground clauses.

Can we also use SAT solver to simplify non-ground clauses? Yes!

Non-ground simplifications by SAT/SMT *(K. IJCAR'08)*

Off-the-shelf SAT solver can be used to simplify ground clauses.

Can we also use SAT solver to simplify non-ground clauses? Yes!

The main idea:

$S_{gr} \models \forall \bar{x} C(\bar{x})$

Non-ground simplifications by SAT/SMT *(K. IJCAR'08)*

Off-the-shelf SAT solver can be used to simplify ground clauses.

Can we also use SAT solver to simplify non-ground clauses? Yes!

The main idea:

$S_{gr} \models \forall \bar{x} C(\bar{x})$ $\qquad\qquad$ $S_{gr} \models C(\bar{d})$ for fresh $\bar{d}$

Non-ground simplifications by SAT/SMT *(K. IJCAR'08)*

Off-the-shelf SAT solver can be used to simplify ground clauses.

Can we also use SAT solver to simplify non-ground clauses? Yes!

The main idea:

$S_{gr} \models \forall \bar{x} C(\bar{x})$	$S_{gr} \models C(\bar{d})$ for fresh $\bar{d}$
$C_1(\bar{x}), \ldots, C_n(\bar{x}) \in S$	$C_1(\bar{d}), \ldots, C_n(\bar{d}) \models C(\bar{d})$

Non-ground simplifications by SAT/SMT *(K. IJCAR'08)*

Off-the-shelf SAT solver can be used to simplify ground clauses.

Can we also use SAT solver to simplify non-ground clauses? Yes!

The main idea:

$$S_{gr} \models \forall \bar{x}\, C(\bar{x}) \qquad\qquad S_{gr} \models C(\bar{d}) \quad \text{for fresh} \quad \bar{d}$$

$$C_1(\bar{x}), \ldots, C_n(\bar{x}) \in S \qquad\qquad C_1(\bar{d}), \ldots, C_n(\bar{d}) \models C(\bar{d}) \text{ as}$$

$$C_1(\bar{x}), \ldots, C_n(\bar{x}) \prec C(\bar{x}) \qquad\qquad \text{in Global Subsumption}$$

Non-Ground Global Subsumption

Non-Ground Global Subsumption

S	C
$\neg P(x) \vee Q(x)$	$S(x, y) \vee Q(x)$
$\neg Q(x) \vee S(x, y)$	
$P(x) \vee S(x, y)$	

Simplify first-order by purely ground reasoning!

Non-Ground Global Subsumption

$$\frac{S}{\begin{array}{l}\neg P(x) \lor Q(x)\\ \neg Q(x) \lor S(x,y)\\ P(x) \lor S(x,y)\end{array}}$$

$$\frac{C}{S(x,y) \lor Q(x)}$$

$$\frac{S_{gr}}{\begin{array}{l}\neg P(a) \lor Q(a)\\ \neg Q(a) \lor S(a,b)\\ P(a) \lor S(a,b)\end{array}}$$

$$\frac{C_{gr}}{S(a,b) \lor Q(a)}$$

Simplify first-order by purely ground reasoning!

Non-Ground Global Subsumption

$$\frac{S}{\begin{array}{l}\neg P(x) \lor Q(x) \\ \neg Q(x) \lor S(x,y) \\ P(x) \lor S(x,y)\end{array}} \qquad \frac{C}{S(x,y) \lor Q(x)}$$

$$\frac{S_{gr}}{\begin{array}{l}\neg P(a) \lor Q(a) \\ \neg Q(a) \lor S(a,b) \\ P(a) \lor S(a,b)\end{array}} \qquad \frac{C_{gr}}{S(a,b) \lor \cancel{Q(a)}}$$

Simplify first-order by purely ground reasoning!

Non-Ground Global Subsumption

$$\frac{S}{\begin{array}{l}\neg P(x) \vee Q(x)\\ \neg Q(x) \vee S(x,y)\\ P(x) \vee S(x,y)\end{array}}$$

$$\frac{C}{S(x,y) \vee \cancel{Q(x)}}$$

$$\frac{S_{gr}}{\begin{array}{l}\neg P(a) \vee Q(a)\\ \neg Q(a) \vee S(a,b)\\ P(a) \vee S(a,b)\end{array}}$$

$$\frac{C_{gr}}{S(a,b) \vee \cancel{Q(a)}}$$

Simplify first-order by purely ground reasoning!

Non-Ground Global Subsumption

S	C
$\neg P(x) \vee Q(x)$	$S(x, y) \vee \cancel{Q(x)}$
$\cancel{\neg Q(x) \vee S(x, y)}$	
$\cancel{P(x) \vee S(x, y)}$	

S_{gr}	C_{gr}
$\neg P(a) \vee Q(a)$	$S(a, b) \vee \cancel{Q(a)}$
$\cancel{\neg Q(a) \vee S(a, b)}$	
$\cancel{P(a) \vee S(a, b)}$	

Simplify first-order by purely ground reasoning!

Inst-Gen summary

Inst-Gen modular instantiation based reasoning for first-order logic.

- Inst-Gen combines efficient ground reasoning with first-order reasoning
- sound and complete for first-order logic
- decision procedure for effectively propositional logic (EPR)
- redundancy elimination
  - strict subsumption, subsumption resolution
  - global subsumption:

    non-ground simplifications using SAT/SMT reasoning
  - dismatching constraints
  - preprocessing:
    - predicate elimination
    - sort inference: EPR and non-cyclic sorts
    - semantic filter
    - definition inference

Equational instantiation-based reasoning

Equality and Paramodulation

Superposition calculus:

$$\frac{C \vee s \simeq t \quad L[s'] \vee D}{(C \vee D \vee L[t])\theta}$$

where (i) $\theta = \mathrm{mgu}(s, s')$, (ii) s' is not a variable, (iii) $s\theta\sigma \succ t\theta\sigma$, (iv) . . .

The same weaknesses as resolution has:

- Inefficient in the ground/EPR case
- Length of clauses can grow fast
- Recombination of clauses
- No explicit model representation

Equality Superposition vs Inst-Gen

Superposition

$$\frac{C \vee l \simeq r \quad L[l'] \vee D}{(C \vee D \vee L[r])\theta}$$

$$\theta = \mathrm{mgu}(l, l')$$

Instantiation?

$$\frac{C \vee l \simeq r \qquad L[l'] \vee D}{(C \vee l \simeq r)\theta \quad (L[l'] \vee D)\theta}$$

$$\theta = \mathrm{mgu}(l, l')$$

Equality Superposition vs Inst-Gen

Superposition

$$\frac{C \vee l \simeq r \quad L[l'] \vee D}{(C \vee D \vee L[r])\theta}$$

$$\theta = \mathrm{mgu}(l, l')$$

Instantiation?

$$\frac{C \vee l \simeq r \qquad L[l'] \vee D}{(C \vee l \simeq r)\theta \quad (L[l'] \vee D)\theta}$$

$$\theta = \mathrm{mgu}(l, l')$$

Incomplete !

Superposition+Instantiation

$$\begin{array}{ccc} f(h(y)) & \simeq & c \\ h(x) & \simeq & x \\ f(a) & \not\simeq & c \end{array}$$

This set is inconsistent but the contradiction is not deducible by the inference system above.

Superposition+Instantiation

$$\begin{array}{ccc} f(h(y)) & \simeq & c \\ h(x) & \simeq & x \\ f(a) & \not\simeq & c \end{array}$$

This set is inconsistent but the contradiction is not deducible by the inference system above.

The idea is to consider proofs generated by unit superposition:

$$\cfrac{\cfrac{\cfrac{h(x) \simeq x \quad f(h(y)) \simeq c}{f(x) \simeq c} \qquad f(a) \not\simeq c}{c \not\simeq c}}{\square}$$

Superposition+Instantiation

$$\begin{array}{ccc} f(h(y)) & \simeq & c \\ h(x) & \simeq & x \\ f(a) & \not\simeq & c \end{array}$$

This set is inconsistent but the contradiction is not deducible by the inference system above.

The idea is to consider proofs generated by unit superposition:

$$\cfrac{\cfrac{h(x) \simeq x \quad f(h(y)) \simeq c}{f(x) \simeq c}\ [x/y] \qquad f(a) \not\simeq c}{\cfrac{c \not\simeq c}{\square}}\ [a/x]$$

Superposition+Instantiation

$$\begin{array}{ccc} f(h(y)) & \simeq & c \\ h(x) & \simeq & x \\ f(a) & \not\simeq & c \end{array}$$

This set is inconsistent but the contradiction is not deducible by the inference system above.

The idea is to consider proofs generated by unit superposition:

$$\dfrac{\dfrac{\dfrac{h(x) \simeq x \quad f(h(y)) \simeq c}{f(x) \simeq c}\,[x/y] \qquad f(a) \not\simeq c}{c \not\simeq c}\,[a/x]}{\square}$$

Propagating substitutions: $\{h(a) \simeq a; f(h(a)) \simeq c; f(a) \not\simeq c\}$ ground unsatisfiable.

Superposition+Instantiation

$$\begin{array}{cccccc} f(h(y)) & \simeq & c & \vee & C_1(y,u) \\ h(x) & \simeq & x & \vee & C_2(x,v) \\ f(a) & \not\simeq & c & \vee & C_3(e) \end{array}$$

This set is inconsistent but the contradiction is not deducible by the inference system above.

The idea is to consider proofs generated by unit superposition:

$$\dfrac{\dfrac{\dfrac{h(x) \simeq x \quad f(h(y)) \simeq c}{f(x) \simeq c}\,[x/y] \qquad f(a) \not\simeq c}{c \not\simeq c}\,[a/x]}{\square}$$

Propagating substitutions: $\{h(a) \simeq a; f(h(a)) \simeq c; f(a) \not\simeq c\}$
ground unsatisfiable.

Superposition+Instantiation

$$\begin{array}{ccccc} f(h(y)) & \simeq & c & \vee & C_1(y,u) \\ h(x) & \simeq & x & \vee & C_2(x,v) \\ f(a) & \not\simeq & c & \vee & C_3(e) \end{array} \qquad \begin{array}{ccccc} f(h(a)) & \simeq & c & \vee & C_1(a,u) \\ h(a) & \simeq & a & \vee & C_2(a,v) \\ f(a) & \not\simeq & c & \vee & C_3(e) \end{array}$$

This set is inconsistent but the contradiction is not deducible by the inference system above.

The idea is to consider proofs generated by unit superposition:

$$\cfrac{\cfrac{\cfrac{h(x) \simeq x \quad f(h(y)) \simeq c}{f(x) \simeq c}\,[x/y] \qquad f(a) \not\simeq c}{c \not\simeq c}\,[a/x]}{\square}$$

Propagating substitutions: $\{h(a) \simeq a; f(h(a)) \simeq c; f(a) \not\simeq c\}$
ground unsatisfiable.

Inst-Gen-Eq instantiation-based equational reasoning

$$\boxed{\begin{array}{c}\text{f.-o. clauses}\\ S\end{array}}$$

Inst-Gen-Eq instantiation-based equational reasoning

$$\boxed{\begin{array}{c}\text{f.-o. clauses}\\ S\end{array}} \xrightarrow{\perp\,:\,\bar{x}\,\to\,\perp} \boxed{\begin{array}{c}\text{Ground Clauses}\\ S_{\perp}\end{array}}$$

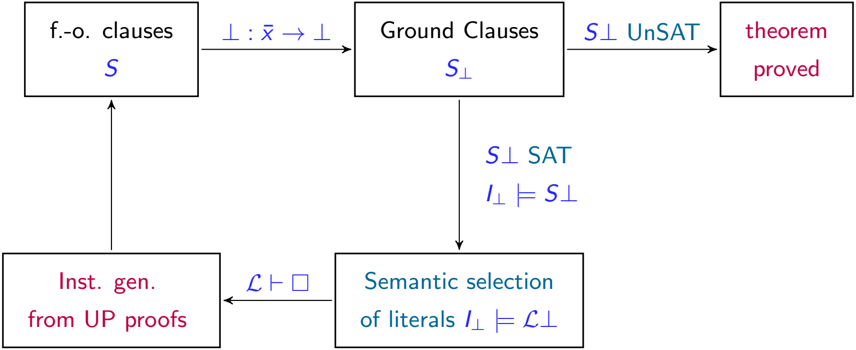
Inst-Gen-Eq instantiation-based equational reasoning

$$\begin{tikzcd}[column sep=huge] \boxed{\begin{array}{c}\text{f.-o. clauses}\\ S\end{array}} \arrow[r, "\bot : \bar{x} \to \bot"] & \boxed{\begin{array}{c}\text{Ground Clauses}\\ S\bot\end{array}} \arrow[r, "S\bot \text{ UnSAT}"] & \boxed{\begin{array}{c}\text{theorem}\\ \text{proved}\end{array}} \end{tikzcd}$$

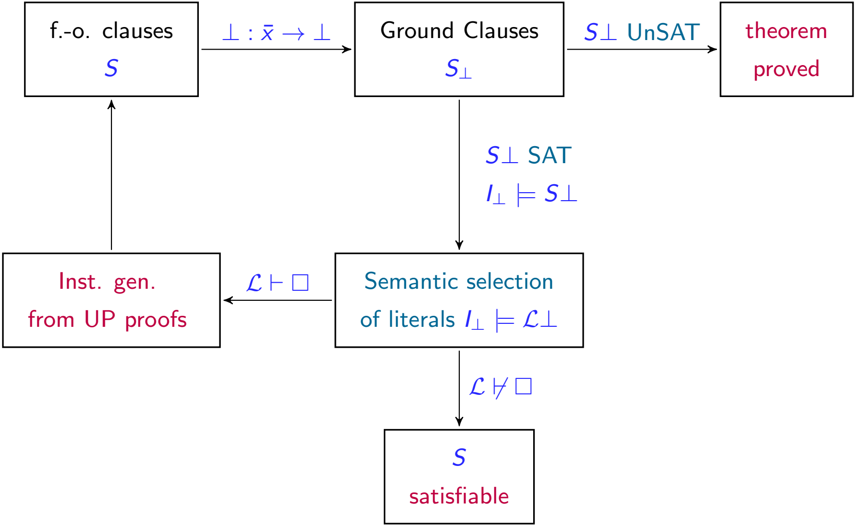
Inst-Gen-Eq instantiation-based equational reasoning

$$\begin{tikzcd}
\boxed{\begin{matrix}\text{f.-o. clauses} \\ S\end{matrix}} \arrow[r, "\bot : \bar{x} \to \bot"] & \boxed{\begin{matrix}\text{Ground Clauses} \\ S_\bot\end{matrix}} \arrow[r, "S\bot \text{ UnSAT}"] \arrow[d, "\begin{matrix} S\bot \text{ SAT} \\ I_\bot \models S\bot \end{matrix}"] & \boxed{\begin{matrix}\text{theorem} \\ \text{proved}\end{matrix}} \\
& \boxed{\begin{matrix}\text{Semantic selection} \\ \text{of literals } I_\bot \models \mathcal{L}\bot\end{matrix}} &
\end{tikzcd}$$

Inst-Gen-Eq instantiation-based equational reasoning



Inst-Gen-Eq instantiation-based equational reasoning



Theorem. Inst-Gen-Eq is sound and complete.

Inst-Gen-Eq: Key properties

Inst-Gen-Eq:

- combines SMT for ground reasoning and superposition-based unit reasoning
- sound and complete for first-order logic with equality
- unit superposition does not have weaknesses of the general superposition
- all redundancy elimination techniques from Inst-Gen are applicable to Inst-Gen-Eq
- redundancy elimination become more powerful: now we can use SMT to simplify first-order rather than SAT

Theory instantiation

Theory instantiation

$\boxed{\begin{array}{c}\text{f.-o. clauses } S \\ \text{theory } T\end{array}}$

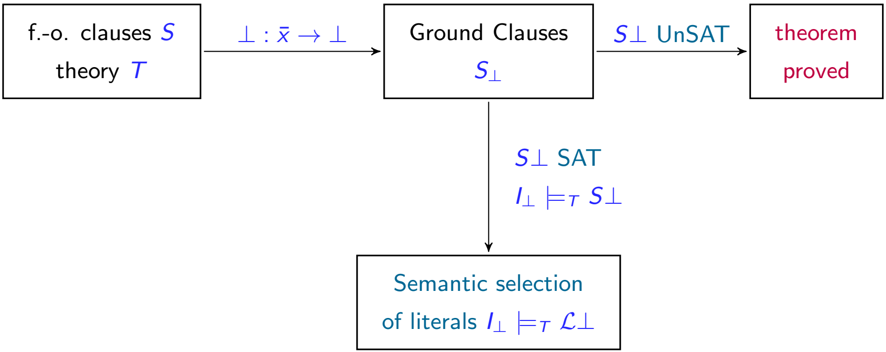
Theory instantiation

$$\boxed{\begin{array}{c}\text{f.-o. clauses } S \\ \text{theory } T\end{array}} \xrightarrow{\perp : \bar{x} \to \perp} \boxed{\begin{array}{c}\text{Ground Clauses} \\ S_{\perp}\end{array}}$$

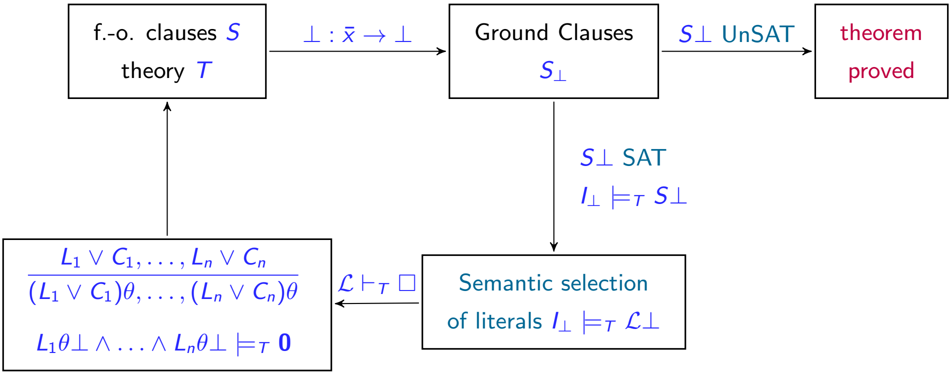
Theory instantiation

$$\boxed{\begin{array}{c}\text{f.-o. clauses } S\\ \text{theory } T\end{array}} \xrightarrow{\;\bot \,:\, \bar{x} \to \bot\;} \boxed{\begin{array}{c}\text{Ground Clauses}\\ S_{\bot}\end{array}} \xrightarrow{\;S\bot \text{ UnSAT}\;} \boxed{\begin{array}{c}\text{theorem}\\ \text{proved}\end{array}}$$

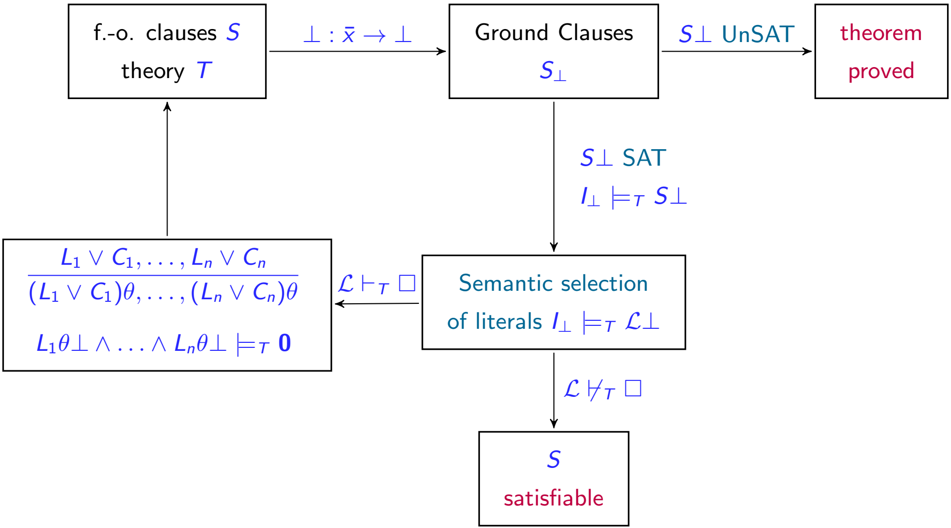
Theory instantiation



Theory instantiation



Theory instantiation



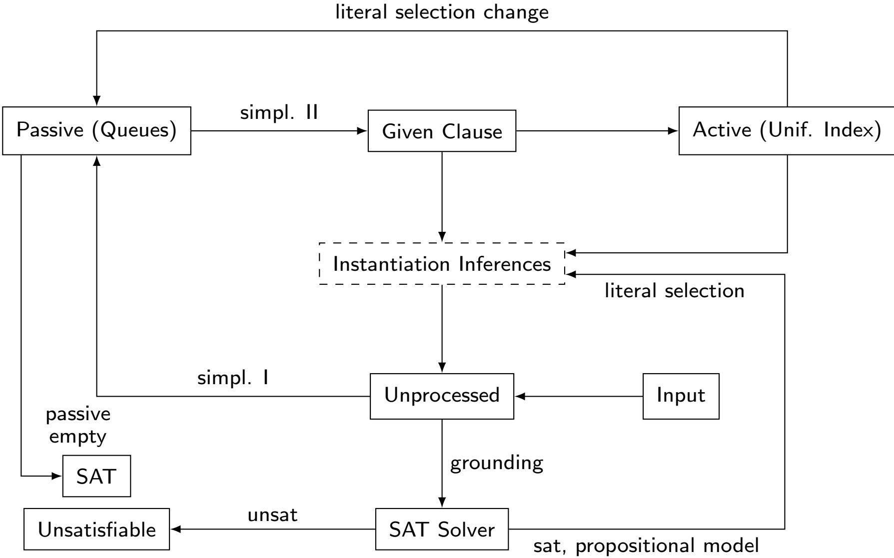
Implementation

iProver general features

iProver an instantiation-based theorem prover for FOL based on Inst-Gen.

- Proof search guided by SAT solver
- Redundancy elimination global subsumption, dismatching constraints, predicate elimination, semantic filtering, splitting...
- Indexing techniques for inferences and simplifications
- Sort inference, non-cyclic sorts
- Combination with resolution
- Finite model finding based on EPR/sort inference/non-cyclic sorts
- Bounded model checking and k-induction
- QBF and bit-vectors
- Planning
- Query answering
- Proof representation: non-trivial due to global solver simplifications
- Model representation: using definitional extensions

Inst-Gen Loop



CASC 2018

EPR:

	iProver	Vampire	E	LEO-III
prob solved	133	128	27	17

First-order SAT:

	Vampire	iProver	CVC4	E
prob solved	191	137	116	38

Applications and the EPR fragment

Effectively Propositional Logic (EPR)

EPR: $\exists^*\forall^*$ fragment of first-order logic

EPR after Skolemization: No functions except constants

$$P(x, y, d) \vee \neg Q(c, y, x)$$

Effectively Propositional Logic (EPR)

EPR: $\exists^*\forall^*$ fragment of first-order logic

EPR after Skolemization: No functions except constants

$$P(x, y, d) \vee \neg Q(c, y, x)$$

Transitivity: $\neg P(x, y) \vee \neg P(y, z) \vee P(x, z)$

Symmetry: $P(x, y) \vee \neg P(y, x)$

Verification:

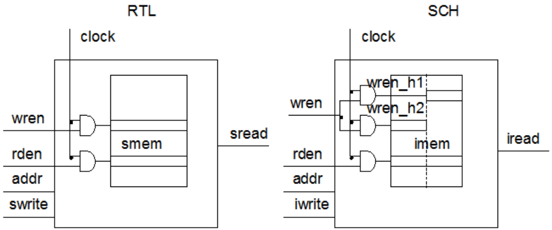
$$\begin{aligned} &\forall A(\mathtt{wren}_{h1} \wedge A = \mathtt{wraddrFunc} \rightarrow \\ &\qquad \forall B(\mathtt{range}_{[35,0]}(B) \rightarrow (\mathtt{imem}'(A, B) \leftrightarrow \mathtt{iwrite}(B)))). \end{aligned}$$

Applications:

- Hardware verification: bounded model checking/bit-vectors
- Program verification: linked data structures (Sagiv)
- Planning/Scheduling
- Knowledge representation
- Finite model finding

EPR is hard for resolution, but decidable by instantiation methods.

Hardware verification



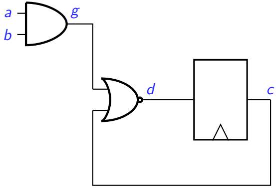
Functional Equivalence Checking

- The same functional behaviour can be implemented in different ways
- Optimised for:
  - Timing – better performance
  - Power – longer battery life
  - Area – smaller chips
- Verification: optimisations do not change functional behaviour

Method of choice: Bounded Model Checking (BMC)

Biere, Cimatti, Clarke, Zhu (TACAS'99)

SAT-based bounded model checking



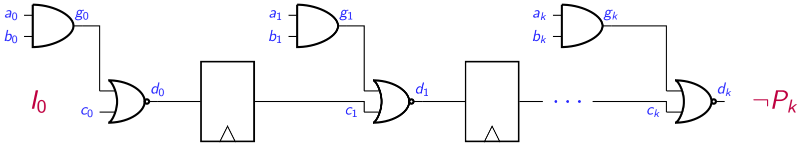
Symbolic representation:

$$
\begin{aligned}
I = \quad & (a_0 \leftrightarrow \neg c_0) \wedge (c_0 \rightarrow b_0) \\
& (g_0 \leftrightarrow a_0 \wedge b_0) \wedge (d_0 \leftrightarrow \neg g_0 \wedge \neg c_0)
\end{aligned}
$$

$$
\begin{aligned}
T = \quad & \\
& a' \leftrightarrow a & \wedge \\
& b' \leftrightarrow b & \wedge \\
& g' \leftrightarrow a' \wedge b' & \wedge \\
& c' \leftrightarrow d & \wedge \\
& d' \leftrightarrow \neg c' \wedge \neg g'
\end{aligned}
$$

$$
P = \quad (d \leftrightarrow \neg g)
$$

SAT-based bounded model checking (unrolling)



The system is unsafe if and only if

$$I_0 \wedge T_{<1,2>} \wedge \ldots \wedge T_{<k-1,k>} \wedge \neg P_k$$

is satisfiable for some k.

A. Biere, A. Cimatti, E. Clarke, Y. Zhu (TACAS'99)

EPR-based BMC

EPR encoding:

- EPR formulas $F_{init}(S), F_{target}(S), F_{next}(S, S')$
- encoding predicates $init(S)$, $target(S)$, $next(S, S')$

Transition system:

$$\forall S \; [init(S) \rightarrow F_{init}(S)] \tag{1}$$

$$\forall S, S' \; [next(S, S') \rightarrow F_{next}(S, S')] \tag{2}$$

$$\forall S \; [target(S) \leftrightarrow F_{target}(S)] \tag{3}$$

BMC: $init(s_0) \wedge next(s_0, s_1) \wedge \ldots \wedge next(s_{n-1}, s_n) \wedge \neg target(s_n)$

EPR-based BMC

EPR encoding:

- EPR formulas $F_{init}(S), F_{target}(S), F_{next}(S, S')$
- encoding predicates $init(S)$, $target(S)$, $next(S, S')$

Transition system:

$$\forall S\ [init(S) \rightarrow F_{init}(S)] \tag{1}$$

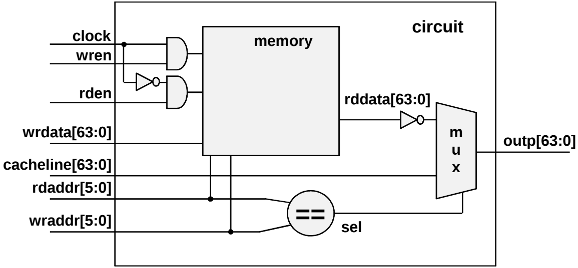
$$\forall S, S'\ [next(S, S') \rightarrow F_{next}(S, S')] \tag{2}$$

$$\forall S\ [target(S) \leftrightarrow F_{target}(S)] \tag{3}$$

BMC: $init(s_0) \wedge next(s_0, s_1) \wedge \ldots \wedge next(s_{n-1}, s_n) \wedge \neg target(s_n)$

- EPR encoding provides succinct representation
- avoids copying transition relation
- reasoning can be done at higher level
- major challenge: hardware designs are very large and complex

Word level



$$
\begin{array}{l}
\forall \mathtt{S}, \mathtt{S'} (\mathit{next}(\mathtt{S}, \mathtt{S'}) \rightarrow \qquad // \quad \text{write is enabled} \\
\quad \forall y (\mathtt{Assoc}_{\mathtt{wraddr}}(\mathtt{S'}, y) \rightarrow \\
\quad\quad \forall \mathtt{A} (\mathtt{clock}(\mathtt{S'}) \wedge \mathtt{wren}(\mathtt{S'}) \wedge \mathtt{A} = y \rightarrow \\
\quad\quad\quad \forall \mathtt{B} (\mathtt{range}_{[0,63]}(\mathtt{B}) \rightarrow (\mathtt{mem}(\mathtt{S'}, \mathtt{A}, \mathtt{B}) \leftrightarrow \mathtt{wrdata}(\mathtt{S}, \mathtt{B})))))).
\end{array}
$$

BMC with memories and bit-vectors

first-order predicates: $\mathtt{mem}(\mathtt{S}, \mathtt{A}, \mathtt{B}), \mathtt{wrdata}(\mathtt{S}, \mathtt{B})$.

M. Emmer, Z. Khasidashvili, K. Korovin, C. Sticksel, A. Voronkov IJCAR'12

Properties of EPR

Direct reduction to SAT — exponential blow-up.

Satisfiability for EPR is NEXPTIME-complete.

More succinct but harder to solve.... Any gain?

Properties of EPR

Direct reduction to SAT — exponential blow-up.

Satisfiability for EPR is NEXPTIME-complete.

More succinct but harder to solve.... Any gain?

Yes: Reasoning can be done at a more general level.

Restricting instances:

$$\neg\texttt{mem}(a_1, x_1) \vee \neg\texttt{mem}(a_2, x_2) \vee \ldots \neg\texttt{mem}(a_n, x_n)$$

$$\texttt{mem}(b_1, x_1) \vee \texttt{mem}(b_2, x_2) \vee \ldots \vee \texttt{mem}(b_n, x_n)$$

Properties of EPR

Direct reduction to SAT — exponential blow-up.

Satisfiability for EPR is NEXPTIME-complete.

More succinct but harder to solve.... Any gain?

Yes: Reasoning can be done at a more general level.

Restricting instances:

$$\neg\texttt{mem}(a_1, x_1) \vee \neg\texttt{mem}(a_2, x_2) \vee \ldots \neg\texttt{mem}(a_n, x_n)$$

$$\texttt{mem}(b_1, x_1) \vee \texttt{mem}(b_2, x_2) \vee \ldots \vee \texttt{mem}(b_n, x_n)$$

General lemmas:

$$\neg bv_1(x) \vee bv_2(x) \qquad \neg bv_2(x) \vee \texttt{mem}(x, y)$$

$$bv_1(x) \vee \texttt{mem}(x, y)$$

Properties of EPR

Direct reduction to SAT — exponential blow-up.

Satisfiability for EPR is NEXPTIME-complete.

More succinct but harder to solve.... Any gain?

Yes: Reasoning can be done at a more general level.

Restricting instances:

$$\neg\texttt{mem}(a_1, x_1) \vee \neg\texttt{mem}(a_2, x_2) \vee \ldots \neg\texttt{mem}(a_n, x_n)$$

$$\texttt{mem}(b_1, x_1) \vee \texttt{mem}(b_2, x_2) \vee \ldots \vee \texttt{mem}(b_n, x_n)$$

General lemmas:

$$\neg bv_1(x) \vee bv_2(x) \qquad \cancel{\neg bv_2(x) \vee \texttt{mem}(x, y)}$$

$$\cancel{bv_1(x) \vee \texttt{mem}(x, y)} \qquad \texttt{mem}(x, y)$$

Properties of EPR

Direct reduction to SAT — exponential blow-up.

Satisfiability for EPR is NEXPTIME-complete.

More succinct but harder to solve.... Any gain?

Yes: Reasoning can be done at a more general level.

Restricting instances:

$$\neg\mathtt{mem}(a_1, x_1) \vee \neg\mathtt{mem}(a_2, x_2) \vee \ldots \neg\mathtt{mem}(a_n, x_n)$$

$$\mathtt{mem}(b_1, x_1) \vee \mathtt{mem}(b_2, x_2) \vee \ldots \vee \mathtt{mem}(b_n, x_n)$$

General lemmas:

$$\neg bv_1(x) \vee bv_2(x) \qquad \cancel{\neg bv_2(x) \vee \mathtt{mem}(x, y)}$$

$$\cancel{bv_1(x) \vee \mathtt{mem}(x, y)} \qquad \mathtt{mem}(x, y)$$

Properties of EPR

Direct reduction to SAT — exponential blow-up.

Satisfiability for EPR is NEXPTIME-complete.

More succinct but harder to solve.... Any gain?

Yes: Reasoning can be done at a more general level.

Restricting instances:

$$\neg\texttt{mem}(a_1, x_1) \vee \neg\texttt{mem}(a_2, x_2) \vee \ldots \neg\texttt{mem}(a_n, x_n)$$
$$\texttt{mem}(b_1, x_1) \vee \texttt{mem}(b_2, x_2) \vee \ldots \vee \texttt{mem}(b_n, x_n)$$

General lemmas:

$$\begin{array}{cc} \neg bv_1(x) \vee bv_2(x) & \cancel{\neg bv_2(x) \vee \texttt{mem}(x, y)} \\ \cancel{bv_1(x) \vee \texttt{mem}(x, y)} & \texttt{mem}(x, y) \end{array}$$

Quantified invariants:

$$\forall s \forall x\ [cond(s, x) \rightarrow prop(s, x)]$$

Properties of EPR

Direct reduction to SAT — exponential blow-up.

Satisfiability for EPR is NEXPTIME-complete.

More succinct but harder to solve.... Any gain?

Yes: Reasoning can be done at a more general level.

Restricting instances:

$$\neg\mathtt{mem}(a_1, x_1) \vee \neg\mathtt{mem}(a_2, x_2) \vee \ldots \neg\mathtt{mem}(a_n, x_n)$$
$$\mathtt{mem}(b_1, x_1) \vee \mathtt{mem}(b_2, x_2) \vee \ldots \vee \mathtt{mem}(b_n, x_n)$$

General lemmas:

$$\neg bv_1(x) \vee bv_2(x) \qquad \cancel{\neg bv_2(x) \vee \mathtt{mem}(x, y)}$$
$$\cancel{bv_1(x) \vee \mathtt{mem}(x, y)} \qquad \mathtt{mem}(x, y)$$

Quantified invariants:

$$\forall s \forall x\ [cond(s, x) \rightarrow prop(s, x)]$$

Using more expressive logics can speed up reasoning!

Experiments: iProver vs Intel BMC

Problem	# Memories	# Transient BVs	Intel BMC	iProver BMC
ROB2	2 (4704 bits)	255 (3479 bits)	50	8
DCC2	4 (8960 bits)	426 (1844 bits)	8	11
DCC1	4 (8960 bits)	1827 (5294 bits)	7	8
DCI1	32 (9216 bits)	3625 (6496 bits)	6	4
BPB2	4 (10240 bits)	550 (4955 bits)	50	11
SCD2	2 (16384 bits)	80 (756 bits)	4	14
SCD1	2 (16384 bits)	556 (1923 bits)	4	12
PMS1	8 (46080 bits)	1486 (6109 bits)	2	10

Large memories:

iProver performs well compared to highly optimised Intel SAT-based model checker.

From bounded to unbounded model checking

EPR-based k-induction

EPR-based k-induction

Base case:

$$init(s_0) \wedge target(s_0) \wedge next(s_0, s_1) \wedge \ldots \wedge next(s_{k-1}, s_k) \wedge \neg target(s_k)$$

Bad states are not reachable in $\leq k$ steps.

Induction case:

$$target(s_0) \wedge next(s_0, s_1) \wedge \ldots \wedge target(s_k) \wedge next(s_n, s_{k+1}) \wedge \neg target(s_{k+1})$$

Assume that bad states are not reachable in $\leq k$ steps then bad states are not reachable in $k+1$ steps.

EPR-based k-induction

Base case:

$$init(s_0) \wedge target(s_0) \wedge next(s_0, s_1) \wedge \ldots \wedge next(s_{k-1}, s_k) \wedge \neg target(s_k)$$

Bad states are not reachable in $\leq k$ steps.

Induction case:

$$target(s_0) \wedge next(s_0, s_1) \wedge \ldots \wedge target(s_k) \wedge next(s_n, s_{k+1}) \wedge \neg target(s_{k+1})$$

Assume that bad states are not reachable in $\leq k$ steps then bad states are not reachable in $k+1$ steps.

Visited states are non-equivalent

$$\forall S, S' \; (S \not\equiv_p S' \rightarrow \exists \bar{x} \; [p(S, \bar{x}) \leftrightarrow \neg p(S', \bar{x})])$$
$$\forall S, S' \; (S \not\equiv_\Sigma S' \rightarrow \bigvee_{p \in \Sigma} S \not\equiv_p S')$$
$$\bigwedge_{0 \leq i \leq j \leq k} s_i \not\equiv_\Sigma s_j$$

QBF to EPR

QBF to EPR

QBF:

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \; [x_1 \vee y_1 \vee \neg y_2 \wedge \ldots]$$

First-order: Domain: $\{\mathbf{1}, \mathbf{0}\}$; $p(\mathbf{1})$; $\neg p(\mathbf{0})$

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \; [p(x_1) \vee p(y_1) \vee \neg p(y_2) \wedge \ldots]$$

Skolemize:

$$\forall x_1 \forall x_2 \; [p(x_1) \vee p(sk_1(x_1)) \vee \neg p(sk_2(x_1, x_2)) \wedge \ldots]$$

EPR: Replace Skolem functions with predicates:

$$\forall x_1 \forall x_2 \; [p(x_1) \vee p_{sk_1}(x_1) \vee \neg p_{sk_2}(x_1, x_2) \wedge \ldots]$$

M. Seidl, F. Lonsing, A. Biere (PAAR'12)

BV with log-encoded width to EPR

BV with log-encoded width to EPR

1	$\cdots$	0	$\cdots$	1	0
2^n		65			0

Encode bit indexes in binary using n bits:

E.g. $\neg bv(\underbrace{0,\ldots,0,1,0,0,0,0,1}_{n})$ represents value 0 at index 65.

Succinct encodings of bit-vector operations avoiding bit-blasting:

bv_and, bv_or, bv_shl, bv_shr, bv_mult, bv_add, $\ldots$.

G. Kovásznai, A. Fröhlich, and A. Biere (CADE'13)

What's next ?

Abstraction refinement reasoning

Large theories in TPTP

TPTP large theories benchmarks:

- Mizar – formalising mathematics
- Isabelle, HOL 4, HOL Light

  translation of higher order problems from different domains into FOL
- CakeML – verification
- Cyc/SUMO – large first-order ontologies

Many of these benchmarks contain hundreds of thousand of axioms.

Large theories in TPTP

TPTP large theories benchmarks:

- Mizar – formalising mathematics
- Isabelle, HOL 4, HOL Light

  translation of higher order problems from different domains into FOL
- CakeML – verification
- Cyc/SUMO – large first-order ontologies

Many of these benchmarks contain hundreds of thousand of axioms.

Large theories in TPTP

TPTP large theories benchmarks:

- Mizar – formalising mathematics
- Isabelle, HOL 4, HOL Light

  translation of higher order problems from different domains into FOL
- CakeML – verification
- Cyc/SUMO – large first-order ontologies

Many of these benchmarks contain hundreds of thousand of axioms.

Observation: large number of axioms is only one indication of complexity.

QBF benchmarks

```
cnf(id_549115,plain,(~$$iProver_qbf_sKE_19800(X0_$i,X1_$i,X2_$i,X3_$i,X4_$i,X5_$i,X6_$i,X7_$i,
X8_$i,X9_$i,X10_$i,X11_$i,X12_$i,X13_$i,X14_$i,X15_$i,X16_$i,X17_$i,X18_$i,X19_$i,X20_$i,X21_$
i,X22_$i,X23_$i,X24_$i,X25_$i,X26_$i,X27_$i,X28_$i,X29_$i,X30_$i,X31_$i,X32_$i,X33_$i,X34_$i,X
35_$i,X36_$i,X37_$i,X38_$i,X39_$i,X40_$i,X41_$i,X42_$i,X43_$i,X44_$i,X45_$i,X46_$i,X47_$i,X48_
$i,X49_$i,X50_$i,X51_$i,X52_$i,X53_$i,X54_$i,X55_$i,X56_$i,X57_$i,X58_$i,X59_$i,X60_$i,X61_$i,
X62_$i,X63_$i,X64_$i,X65_$i,X66_$i,X67_$i,X68_$i,X69_$i,X70_$i,X71_$i,X72_$i,X73_$i,X74_$i,X75
_$i,X76_$i,X77_$i,X78_$i,X79_$i,X80_$i,X81_$i,X82_$i,X83_$i,X84_$i,X85_$i,X86_$i,X87_$i,X88_$i
,X89_$i,X90_$i,X91_$i,X92_$i,X93_$i,X94_$i,X95_$i,X96_$i,X97_$i,X98_$i,X99_$i,X100_$i,X101_$i,
X102_$i,X103_$i,X104_$i,X105_$i,X106_$i,X107_$i,X108_$i,X109_$i,X110_$i,X111_$i,X112_$i,X113_$
i,X114_$i,X115_$i,X116_$i,X117_$i,X118_$i,X119_$i,X120_$i,X121_$i)|$$iProver_qbf_sKE_854)).
cnf(id_549116,plain,(~$$iProver_qbf_sKE_19800(X0_$i,X1_$i,X2_$i,X3_$i,X4_$i,X5_$i,X6_$i,X7_$i,
X8_$i,X9_$i,X10_$i,X11_$i,X12_$i,X13_$i,X14_$i,X15_$i,X16_$i,X17_$i,X18_$i,X19_$i,X20_$i,X21_$
i,X22_$i,X23_$i,X24_$i,X25_$i,X26_$i,X27_$i,X28_$i,X29_$i,X30_$i,X31_$i,X32_$i,X33_$i,X34_$i,X
35_$i,X36_$i,X37_$i,X38_$i,X39_$i,X40_$i,X41_$i,X42_$i,X43_$i,X44_$i,X45_$i,X46_$i,X47_$i,X48_
$i,X49_$i,X50_$i,X51_$i,X52_$i,X53_$i,X54_$i,X55_$i,X56_$i,X57_$i,X58_$i,X59_$i,X60_$i,X61_$i,
X62_$i,X63_$i,X64_$i,X65_$i,X66_$i,X67_$i,X68_$i,X69_$i,X70_$i,X71_$i,X72_$i,X73_$i,X74_$i,X75
_$i,X76_$i,X77_$i,X78_$i,X79_$i,X80_$i,X81_$i,X82_$i,X83_$i,X84_$i,X85_$i,X86_$i,X87_$i,X88_$i
,X89_$i,X90_$i,X91_$i,X92_$i,X93_$i,X94_$i,X95_$i,X96_$i,X97_$i,X98_$i,X99_$i,X100_$i,X101_$i,
X102_$i,X103_$i,X104_$i,X105_$i,X106_$i,X107_$i,X108_$i,X109_$i,X110_$i,X111_$i,X112_$i,X113_$
i,X114_$i,X115_$i,X116_$i,X117_$i,X118_$i,X119_$i,X120_$i,X121_$i)|$$iProver_qbf_sKE_852)).
cnf(id_549117,plain,(~$$iProver_qbf_sKE_9100(X0_$i,X1_$i,X2_$i,X3_$i,X4_$i,X5_$i,X6_$i,X7_$i,X
8_$i,X9_$i,X10_$i,X11_$i,X12_$i,X13_$i,X14_$i,X15_$i,X16_$i,X17_$i,X18_$i,X19_$i,X20_$i,X21_$i
,X22_$i,X23_$i,X24_$i,X25_$i,X26_$i,X27_$i,X28_$i,X29_$i,X30_$i,X31_$i,X32_$i,X33_$i,X34_$i,X3
5_$i,X36_$i,X37_$i,X38_$i,X39_$i,X40_$i,X41_$i,X42_$i,X43_$i,X44_$i,X45_$i,X46_$i,X47_$i,X48_$
i,X49_$i,X50_$i,X51_$i,X52_$i,X53_$i,X54_$i,X55_$i,X56_$i,X57_$i,X58_$i,X59_$i,X60_$i,X61_$i,X
62_$i,X63_$i,X64_$i,X65_$i,X66_$i,X67_$i,X68_$i,X69_$i,X70_$i,X71_$i,X72_$i,X73_$i,X74_$i,X75_
$i,X76_$i,X77_$i,X78_$i,X79_$i,X80_$i,X81_$i,X82_$i,X83_$i,X84_$i,X85_$i,X86_$i,X87_$i,X88_$i,
X89_$i,X90_$i,X91_$i,X92_$i,X93_$i,X94_$i,X95_$i,X96_$i,X97_$i,X98_$i,X99_$i,X100_$i,X101_$i,X
102_$i,X103_$i,X104_$i,X105_$i,X106_$i,X107_$i,X108_$i,X109_$i,X110_$i,X111_$i,X112_$i,X113_$i
,X114_$i,X115_$i,X116_$i,X117_$i,X118_$i,X119_$i,X120_$i,X121_$i)|~$$iProver_qbf_sKE_8843(X0_$
i,X1_$i,X2_$i,X3_$i,X4_$i,X5_$i,X6_$i,X7_$i,X8_$i,X9_$i,X10_$i,X11_$i,X12_$i,X13_$i,X14_$i,X15
_$i,X16_$i,X17_$i,X18_$i,X19_$i,X20_$i,X21_$i,X22_$i,X23_$i,X24_$i,X25_$i,X26_$i,X27_$i,X28_$i
,X29_$i,X30_$i,X31_$i,X32_$i,X33_$i,X34_$i,X35_$i,X36_$i,X37_$i,X38_$i,X39_$i,X40_$i,X41_$i,X4
2_$i,X43_$i,X44_$i,X45_$i,X46_$i,X47_$i,X48_$i,X49_$i,X50_$i,X51_$i,X52_$i,X53_$i,X54_$i,X55_$
i,X56_$i,X57_$i,X58_$i,X59_$i,X60_$i,X61_$i,X62_$i,X63_$i,X64_$i,X65_$i,X66_$i,X67_$i,X68_$i,X
69_$i,X70_$i,X71_$i,X72_$i,X73_$i,X74_$i,X75_$i,X76_$i,X77_$i,X78_$i,X79_$i,X80_$i,X81_$i,X82_
$i,X83_$i,X84_$i,X85_$i,X86_$i,X87_$i,X88_$i,X89_$i,X90_$i,X91_$i,X92_$i,X93_$i,X94_$i,X95_$i,
```

HOL benchmarks

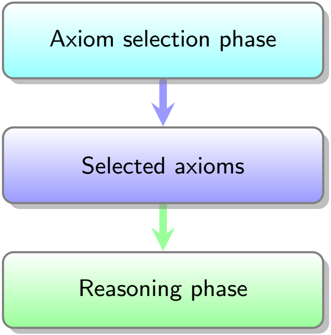
```
fof('thm.misc.read_bytearray_def_compute|split|1',axiom,(
    ! [V_27B_27,V_27A_27,V_27a_27,V_27n_27,V_27get__byte_27] : s('type.option.option'('type.list.list'(V_27A_27))
,'const.misc.read_bytearray_3'(s('type.fcp.cart'(bool,V_27B_27),V_27a_27),s('type.num.num','const.arithmetic.NUME
RAL_1'(s('type.num.num','const.arithmetic.BIT|49|_1'(s('type.num.num',V_27n_27))))),s(fun('type.fcp.cart'(bool,V_
27B_27),'type.option.option'(V_27A_27)),V_27get__byte_27))) = s('type.option.option'('type.list.list'(V_27A_27)),
happ(s(fun(fun(V_27A_27,'type.option.option'('type.list.list'(V_27A_27))),'type.option.option'('type.list.list'(V
_27A_27))),'const.option.option_CASE_2'(s('type.option.option'(V_27A_27),happ(s(fun('type.fcp.cart'(bool,V_27B_27
),'type.option.option'(V_27A_27)),V_27get__byte_27),s('type.fcp.cart'(bool,V_27B_27),V_27a_27))),s('type.option.o
ption'('type.list.list'(V_27A_27)),'const.option.NONE_0'))),s(fun(V_27A_27,'type.option.option'('type.list.list'(
V_27A_27))),'_dst_x0x1_2'(s(fun(fun('type.list.list'(V_27A_27),'type.option.option'('type.list.list'(V_27A_27))),
'type.option.option'('type.list.list'(V_27A_27))),'const.option.option_CASE_2'(s('type.option.option'('type.list.
list'(V_27A_27)),'const.misc.read_bytearray_3'(s('type.fcp.cart'(bool,V_27B_27),'const.words.word_add_2'(s('type.
fcp.cart'(bool,V_27B_27),V_27a_27),s('type.fcp.cart'(bool,V_27B_27),'const.words.n2w_1'(s('type.num.num','const.a
rithmetic.NUMERAL_1'(s('type.num.num','const.arithmetic.BIT|49|_1'(s('type.num.num','const.arithmetic.ZERO_0'))))
))))),s('type.num.num','const.arithmetic.-_2'(s('type.num.num','const.arithmetic.NUMERAL_1'(s('type.num.num','con
st.arithmetic.BIT|49|_1'(s('type.num.num',V_27n_27))))),s('type.num.num','const.arithmetic.NUMERAL_1'(s('type.num
.num','const.arithmetic.BIT|49|_1'(s('type.num.num','const.arithmetic.ZERO_0'))))))),s(fun('type.fcp.cart'(bool,V
_27B_27),'type.option.option'(V_27A_27)),V_27get__byte_27))),s('type.option.option'('type.list.list'(V_27A_27)),'
const.option.NONE_0'))),s(fun(V_27A_27,fun('type.list.list'(V_27A_27),'type.option.option'('type.list.list'(V_27A
_27)))),'_dst_x00x11_2'(s(fun('type.list.list'(V_27A_27),'type.option.option'('type.list.list'(V_27A_27))),'const
.option.SOME_0'),s(fun(V_27A_27,fun('type.list.list'(V_27A_27),'type.list.list'(V_27A_27))),'const.list.CONS_0'))
))))) )).

fof('thm.misc.read_bytearray_def_compute|split|2',axiom,(
    ! [V_27B_27,V_27A_27,V_27a_27,V_27n_27,V_27get__byte_27] : s('type.option.option'('type.list.list'(V_27A_27))
,'const.misc.read_bytearray_3'(s('type.fcp.cart'(bool,V_27B_27),V_27a_27),s('type.num.num','const.arithmetic.NUME
RAL_1'(s('type.num.num','const.arithmetic.BIT2_1'(s('type.num.num',V_27n_27))))),s(fun('type.fcp.cart'(bool,V_27B
_27),'type.option.option'(V_27A_27)),V_27get__byte_27))) = s('type.option.option'('type.list.list'(V_27A_27)),hap
p(s(fun(fun(V_27A_27,'type.option.option'('type.list.list'(V_27A_27))),'type.option.option'('type.list.list'(V_27
A_27))),'const.option.option_CASE_2'(s('type.option.option'(V_27A_27),happ(s(fun('type.fcp.cart'(bool,V_27B_27),'
type.option.option'(V_27A_27)),V_27get__byte_27),s('type.fcp.cart'(bool,V_27B_27),V_27a_27))),s('type.option.opti
on'('type.list.list'(V_27A_27)),'const.option.NONE_0'))),s(fun(V_27A_27,'type.option.option'('type.list.list'(V_2
7A_27))),'_dst_x0x1_2'(s(fun(fun('type.list.list'(V_27A_27),'type.option.option'('type.list.list'(V_27A_27))),'ty
pe.option.option'('type.list.list'(V_27A_27))),'const.option.option_CASE_2'(s('type.option.option'('type.list.lis
t'(V_27A_27)),'const.misc.read_bytearray_3'(s('type.fcp.cart'(bool,V_27B_27),'const.words.word_add_2'(s('type.fcp
.cart'(bool,V_27B_27),V_27a_27),s('type.fcp.cart'(bool,V_27B_27),'const.words.n2w_1'(s('type.num.num','const.arit
hmetic.NUMERAL_1'(s('type.num.num','const.arithmetic.BIT|49|_1'(s('type.num.num','const.arithmetic.ZERO_0')))))))
)),s('type.num.num','const.arithmetic.NUMERAL_1'(s('type.num.num','const.arithmetic.BIT|49|_1'(s('type.num.num',V
_27n_27))))),s(fun('type.fcp.cart'(bool,V_27B_27),'type.option.option'(V_27A_27)),V_27get__byte_27))),s('type.opt
ion.option'('type.list.list'(V_27A_27)),'const.option.NONE_0'))),s(fun(V_27A_27,fun('type.list.list'(V_27A_27),'t
ype.option.option'('type.list.list'(V_27A_27)))),'_dst_x00x11_2'(s(fun('type.list.list'(V_27A_27),'type.option.op
tion'('type.list.list'(V_27A_27))),'const.option.SOME_0'),s(fun(V_27A_27,fun('type.list.list'(V_27A_27),'type.lis
t.list'(V_27A_27))),'const.list.CONS_0'))))))) )).
```

Reasoning with large theories: axiom selection

Previous approaches: select "relevant axioms"

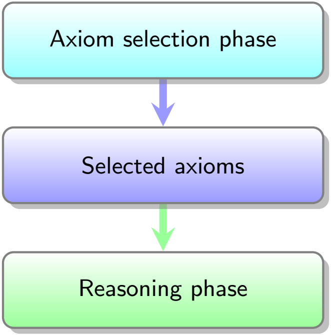
- Semantic or syntactic structure
  - SRASS
  - SInE
- Machine learning
  - MaLARea
- Two phases
  - Axiom selection
  - Reasoning



Reasoning with large theories: axiom selection

Previous approaches: select "relevant axioms"

- Semantic or syntactic structure
  - SRASS
  - SInE
- Machine learning
  - MaLARea
- Two phases
  - Axiom selection
  - Reasoning



Observation: large number of axioms is only one source of complexity.

We also have: large number of arguments; large signatures; long/deep clauses; etc.

Abstraction-refinement approach *L. Hernandez, K. IJCAR'18*

- Abstraction-Refinement
- Interleaving abstraction and reasoning phases

- The abstraction is easier to solve
- If there is no solution, the abstraction is refined

Abstraction-refinement approach *L. Hernandez, K. IJCAR'18*

- Abstraction-Refinement
- Interleaving abstraction and reasoning phases
- Over-Approximation

- The abstraction is easier to solve
- If there is no solution, the abstraction is refined
- If $A \models \bot$ then $\alpha(A) \models \bot$

Abstraction-refinement approach *L. Hernandez, K. IJCAR'18*

- Abstraction-Refinement
- Interleaving abstraction and reasoning phases
- Over-Approximation
- Under-Approximation

- The abstraction is easier to solve
- If there is no solution, the abstraction is refined
- If $A \models \perp$ then $\alpha(A) \models \perp$
- If $\alpha(A) \models \perp$ then $A \models \perp$

Abstraction-refinement approach *L. Hernandez, K. IJCAR'18*

- Abstraction-Refinement
- Interleaving abstraction and reasoning phases
- Over-Approximation
- Under-Approximation
- Combination of approximations

- The abstraction is easier to solve
- If there is no solution, the abstraction is refined
- If $A \models \perp$ then $\alpha(A) \models \perp$
- If $\alpha(A) \models \perp$ then $A \models \perp$
- Converge rapidly to a solution if it exists

Abstraction-Refinement in ATPs

- ...
- Inst-Gen: Ganzinger, Korovin
- SPASS: targeted decidable fragment Teucke, Weidenbach
- Speculative inferences: Bonacina, Lynch, de Moura
- SMT: conflict and model-based instantiation

  de Moura, Ge; Reynolds, Tinelli ...
- AVATAR: new architecture for first-order theorem provers

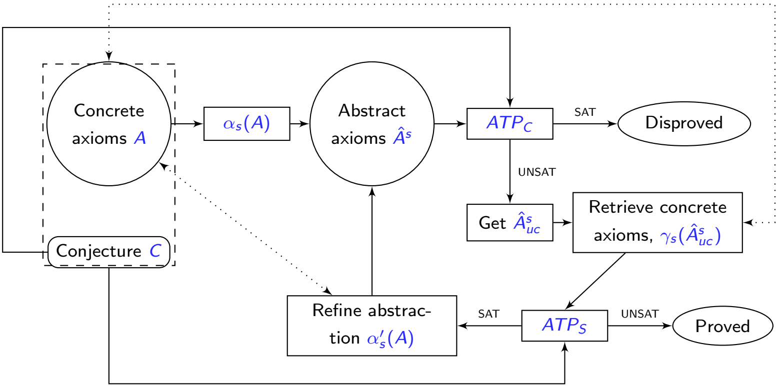
  Voronkov; Reger, Suda, ...

Over-Approximating Abstractions

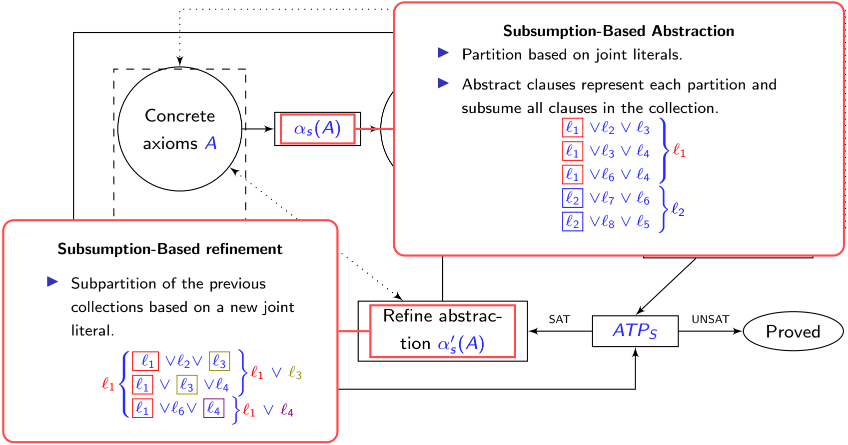
Over-approximation abstractions:

- Subsumption abstraction
- Generalisation abstraction
- Argument filtering abstraction
- Signature grouping abstraction

Over-Approximation Procedure



Over-Approximation Procedure



Subsumption-Based Abstraction

- Partition based on joint literals.
- Abstract clauses represent each partition and subsume all clauses in the collection.

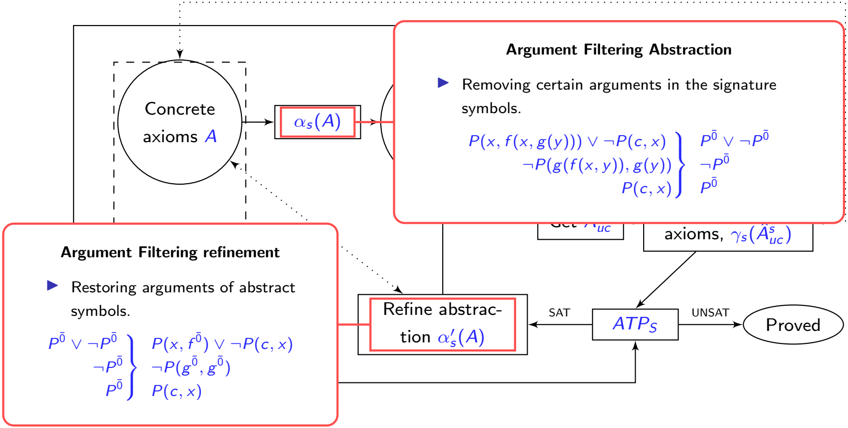
$$\left.\begin{array}{l} \boxed{\ell_1} \vee \ell_2 \vee \ell_3 \\ \boxed{\ell_1} \vee \ell_3 \vee \ell_4 \\ \boxed{\ell_1} \vee \ell_6 \vee \ell_4 \end{array}\right\} \ell_1 \qquad \left.\begin{array}{l} \boxed{\ell_2} \vee \ell_7 \vee \ell_6 \\ \boxed{\ell_2} \vee \ell_8 \vee \ell_5 \end{array}\right\} \ell_2$$

Subsumption-Based refinement

- Subpartition of the previous collections based on a new joint literal.

$$\ell_1 \left\{ \begin{array}{l} \left.\begin{array}{l} \boxed{\ell_1} \vee \ell_2 \vee \boxed{\ell_3} \\ \boxed{\ell_1} \vee \boxed{\ell_3} \vee \ell_4 \end{array}\right\} \ell_1 \vee \ell_3 \\ \left.\begin{array}{l} \boxed{\ell_1} \vee \ell_6 \vee \boxed{\ell_4} \end{array}\right\} \ell_1 \vee \ell_4 \end{array} \right.$$

Over-Approximation Procedure



Argument Filtering Abstraction

- Removing certain arguments in the signature symbols.

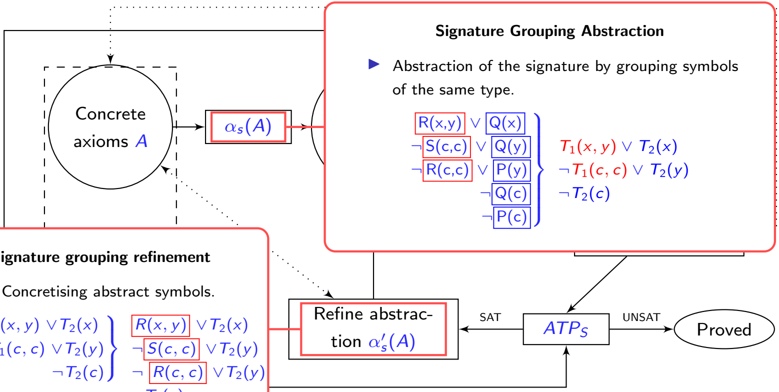
$$\left.\begin{array}{r} P(x, f(x, g(y))) \vee \neg P(c, x) \\ \neg P(g(f(x, y)), g(y)) \\ P(c, x) \end{array}\right\} \begin{array}{l} P^{\bar{0}} \vee \neg P^{\bar{0}} \\ \neg P^{\bar{0}} \\ P^{\bar{0}} \end{array}$$

Argument Filtering refinement

- Restoring arguments of abstract symbols.

$$\left.\begin{array}{r} P^{\bar{0}} \vee \neg P^{\bar{0}} \\ \neg P^{\bar{0}} \\ P^{\bar{0}} \end{array}\right\} \begin{array}{l} P(x, f^{\bar{0}}) \vee \neg P(c, x) \\ \neg P(g^{\bar{0}}, g^{\bar{0}}) \\ P(c, x) \end{array}$$

Over-Approximation Procedure



Signature Grouping Abstraction

- Abstraction of the signature by grouping symbols of the same type.

$$\left.\begin{array}{r} \boxed{R(x,y)} \lor \boxed{Q(x)} \\ \neg\boxed{S(c,c)} \lor \boxed{Q(y)} \\ \neg\boxed{R(c,c)} \lor \boxed{P(y)} \\ \neg\boxed{Q(c)} \\ \neg\boxed{P(c)} \end{array}\right\} \quad \begin{array}{l} T_1(x,y) \lor T_2(x) \\ \neg T_1(c,c) \lor T_2(y) \\ \neg T_2(c) \end{array}$$

Signature grouping refinement

- Concretising abstract symbols.

$$\left.\begin{array}{r} T_1(x,y) \lor T_2(x) \\ \neg\, T_1(c,c) \lor T_2(y) \\ \neg T_2(c) \end{array}\right\} \quad \begin{array}{l} \boxed{R(x,y)} \lor T_2(x) \\ \neg\boxed{S(c,c)} \lor T_2(y) \\ \neg\, \boxed{R(c,c)} \lor T_2(y) \\ \neg T_2(c) \end{array}$$

Generalisation abstraction

- Strengthening abstraction function α_s.
  - Partition axioms $A = \cup_i A_i$; abstract axiom: $\alpha_s(A_i) \models A_i$

$S(f(x))$ $\quad$ $Q(f(x), a) \vee R(g(x))$

$S(h(x, y))$ $\quad$ $Q(z, x) \vee R(x) \vee P(x, z)$

$\neg P(x, h(y, a)) \vee R(y)$ $\quad$ $\neg R(h(f(x), g(y)))$

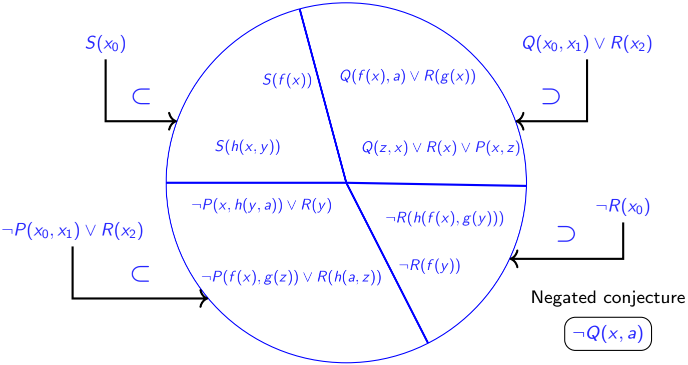
$\neg P(f(x), g(z)) \vee R(h(a, z))$ $\quad$ $\neg R(f(y))$

Negated conjecture

$\boxed{\neg Q(x, a)}$

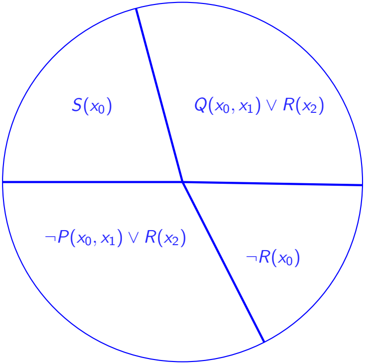
Generalisation abstraction

- Strengthening abstraction function α_s.
  - Partition axioms $A = \cup_i A_i$; abstract axiom: $\alpha_s(A_i) \models A_i$



Generalisation abstraction

- Strengthening abstraction function α_s.
  - Partition axioms $A = \cup_i A_i$; abstract axiom: $\alpha_s(A_i) \models A_i$

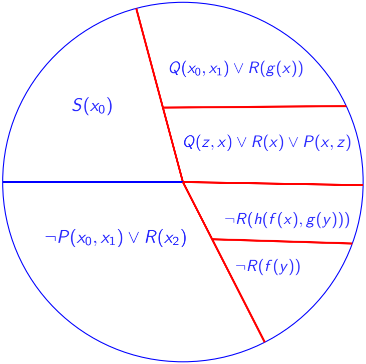


Negated conjecture

$\boxed{\neg Q(x, a)}$

Generalisation abstraction refinement

- Weakening abstraction refinement.
  - Sub-partition groups of concrete axioms involved in an abstract proof.



Negated conjecture

$\boxed{\neg Q(x, a)}$

Generalisation abstraction for termination

Consider the following set of clauses:

$$S = \{ \quad p(\underline{g(x)}, \underline{g(x)}) \vee q(f(\underline{g(x)})) \\ \underline{g(f(f(x)))} \simeq g(f(x)) \}$$

A generalisation abstraction of S:

$$\alpha(S) = \{ \quad p(x, x) \vee q(f(x)) \\ g(f(x)) \simeq g(x) \}$$

Superposition is not applicable after subsumption abstraction and therefore S is satisfiable.

Over-approximation

Over-approximation abstractions:

- Subsumption abstraction
- Generalisation abstraction
- Argument filtering abstraction
- Signature grouping abstraction

Combinations of these abstractions

- `--abstr_ref [sig;subs;arg_filter]`
- abstractions can enable further abstractions: e.g, argument filtering can enable signature grouping which can enable subsumption

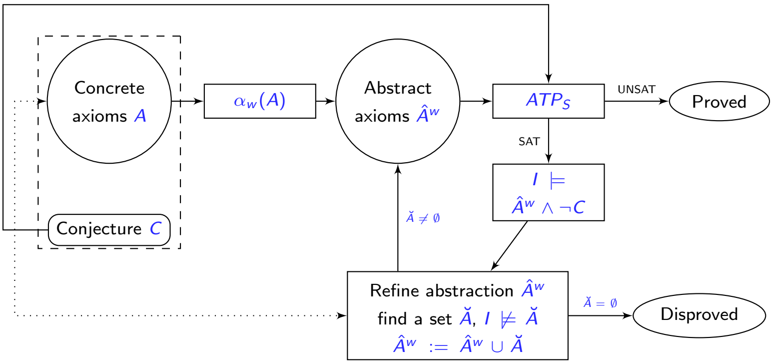
Targeted abstractions:

- abstractions can target fragments e.g., EPR
- block superposition inferences

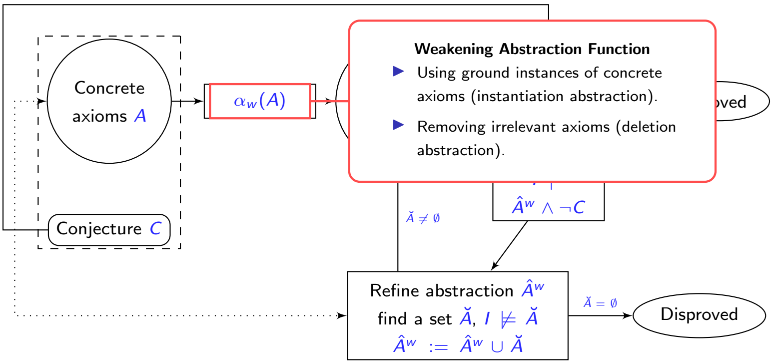
Under-Approximation

- Weakening abstraction function.
  - Removing irrelevant axioms using methods like SInE or MaLARea.
  - Using ground instances of concrete axioms.
- Strengthening abstraction refinement.
  - Turning a model I into a countermodel.
  - Add concrete axioms
  - Generate and add ground instances of axioms

Under-Approximation



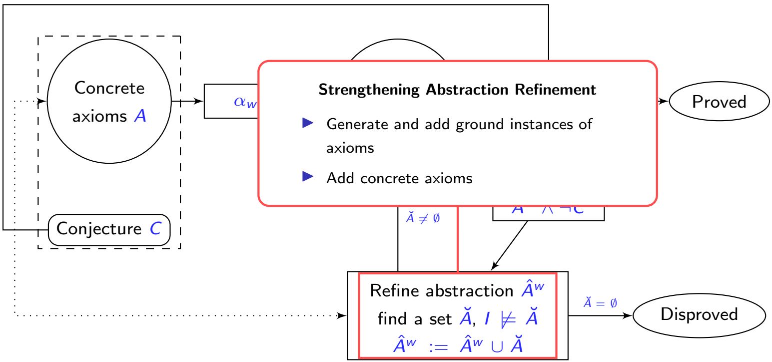
Under-Approximation



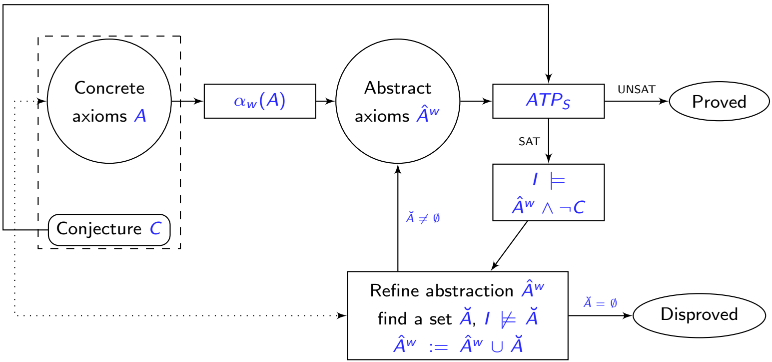
Weakening Abstraction Function

- Using ground instances of concrete axioms (instantiation abstraction).
- Removing irrelevant axioms (deletion abstraction).

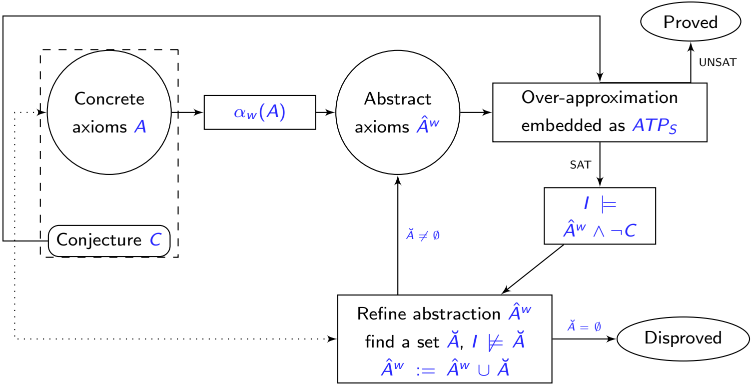
Under-Approximation



Under-Approximation



Combined Approximations



Shared abstractions.

Implementation & Experiments

- Abstraction-refinement implemented in iProver v2.8
- Strategies: *combination* of atomic abstractions

  `--abstr_ref [subs;arg_filger;sig]`

- SInE as under-approximating abstraction

The Most Effective Strategies

Table: SC = Skolem and constant, SS = Skolem and split symb.

Depth	Tolerance	Abstractions	Signature	Arg-filter	Until SAT	Solutions
1	1.0	sig, subs, arg-fil		SS	true	1001
1	2.0	subs, sig, arg-fil	SC		false	42
2	1.0	subs, sig, arg-fil	SC		false	23
1	4.0	arg-fil, sig, subs		SS	true	5
1	1.0	subs, sig, arg-fil	SC	SS	false	4
1	1.0	subs, sig, arg-fil			false	2
2	1.0	sig	SC		false	2
1	8.0	subs, sig, arg-fil			false	2
1	1.0	arg-fil, subs, sig		SS	false	2
2	1.0	arg-fil, sig, subs		SS	true	2
2	1.0	arg-fil			false	1
2	1.0	subs, sig			false	1
					Total	1087

CASC-26

Table: CASC-26 LTB comparison (out of 1500 problems)

Vampire 4.0	Vampire 4.2	MaLARea	iProver 2.8	iProver 2.6	E LTB
1156	1144	1131	1087	777	683

Abstraction-refinement current work

- Abstractions targeted for specific theories
- Goal directed abstractions
- Reuse of abstractions
- Different combination schemes/ ML
- Target abstractions for theories

Conclusions

Instantiation-based theorem proving for first-order logic:

- Modular combination of SAT/SMT and first-order reasoning
- Combination of proof search and model search
- Abstraction-refinement for large/complex problems

Further directions:

- The quest of combining first-order and theories: highly undecidable
- Combination with SMT approaches to quantifier instantiation
- Abstraction-refinement as a generalisation of instantiation based reasoning ?

Extra: efficient datastructures and indexes

Indexing

Why indexing:

- Single subsumption is NP-hard.
- We can have 100,000 clauses in our search space
- Applying naively between all pairs of clauses we need 10,000,000,000 subsumption checks !

Indexing

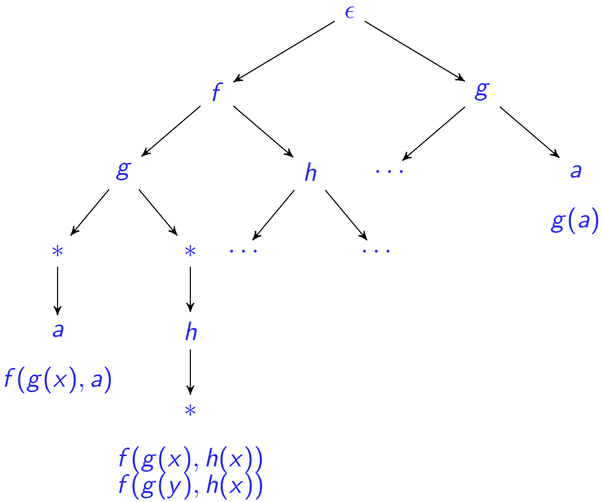
Why indexing:

- Single subsumption is NP-hard.
- We can have 100,000 clauses in our search space
- Applying naively between all pairs of clauses we need 10,000,000,000 subsumption checks !

Indexes in iProver:

- non-perfect discrimination trees for unification, matching
- compressed feature vector indexes for subsumption, subsumption resolution, dismatching constraints.

Unification: Discrimination trees



Efficient filtering unification, matching and generalisation candidates

Subsumption: Feature vector index

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz] works well for subsumption, and many other operations

Design efficient filters based on "features of clauses":

- clause C can not subsume any clause with number of literals strictly less than C

Subsumption: Feature vector index

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz] works well for subsumption, and many other operations

Design efficient filters based on "features of clauses":

- clause C can not subsume any clause with number of literals strictly less than C
- clause C can not subsume any clause with number of positive literals strictly less than C

Subsumption: Feature vector index

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz] works well for subsumption, and many other operations

Design efficient filters based on "features of clauses":

- clause C can not subsume any clause with number of literals strictly less than C
- clause C can not subsume any clause with number of positive literals strictly less than C
- clause C can not subsume any clause with the number of occurrences of a symbol f less than in C

Subsumption: Feature vector index

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz] works well for subsumption, and many other operations

Design efficient filters based on “features of clauses”:

- clause C can not subsume any clause with number of literals strictly less than C
- clause C can not subsume any clause with number of positive literals strictly less than C
- clause C can not subsume any clause with the number of occurrences of a symbol f less than in C
- ...

Subsumption: Feature vector index

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz] works well for subsumption, and many other operations

Design efficient filters based on "features of clauses":

- clause C can not subsume any clause with number of literals strictly less than C
- clause C can not subsume any clause with number of positive literals strictly less than C
- clause C can not subsume any clause with the number of occurrences of a symbol f less than in C
- ...

Subsumption: Feature vector index

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz] works well for subsumption, and many other operations

Design efficient filters based on “features of clauses”:

- clause C can not subsume any clause with number of literals strictly less than C
- clause C can not subsume any clause with number of positive literals strictly less than C
- clause C can not subsume any clause with the number of occurrences of a symbol f less than in C
- ...

Feature vector index

Fix: a list of features:

1. number of literals
2. number of occurrences of f
3. number of occurrences of g

With each clause associate a feature vector:

numeric vector of feature values

Example: feature vector of $C = p(f(f(x))) \lor \neg p(g(y))$ is

$fv(C) = [2, 2, 1]$

Arrange feature vectors in a trie data structure similar to discrimination tree

Feature vector index

Fix: a list of features:

1. number of literals
2. number of occurrences of f
3. number of occurrences of g

With each clause associate a feature vector:

numeric vector of feature values

Example: feature vector of $C = p(f(f(x))) \lor \neg p(g(y))$ is

$fv(C) = [2, 2, 1]$

Arrange feature vectors in a trie data structure similar to discrimination tree

For retrieving all candidates which can be subsumed by C we need to traverse only vectors which are component-wise greater or equal to $fv(C)$.

Compressed feature vector index [iProver]

The signature based features are most useful but also expensive.

Example: is signature contains 1000 symbols and we use all symbols as features then feature vector for every clause will be 1000 in length.

Compressed feature vector index *[iProver]*

The signature based features are most useful but also expensive.

Example: is signature contains 1000 symbols and we use all symbols as features then feature vector for every clause will be 1000 in length.

Basic idea: for each clause most features will be 0.

Compressed feature vector index *[iProver]*

The signature based features are most useful but also expensive.

Example: is signature contains 1000 symbols and we use all symbols as features then feature vector for every clause will be 1000 in length.

Basic idea: for each clause most features will be 0.

Compress feature vector: use list of pairs $[(p_1, v_1), \ldots, (p_n, v_1)]$ where p_i are non-zero positions and v_i are values that start from this position. Sequential positions with the same value are combined.

iProver uses compressed feature vector index for forward and backward subsumption, subsumption resolution and dismatching constraints.