Automated reasoning for first-order logic

Theory, Practice and Challenges

Konstantin Korovin[1]

The University of Manchester
UK

korovin@cs.man.ac.uk

Part II

[1]supported by a Royal Society University Fellowship

Modular instantiation-based reasoning

SAT/SMT vs First-Order

The problem: Show that a given formula is a theorem.

Ground (SAT/SMT)

$$P(a) \lor Q(c, d)$$
$$\neg P(a) \lor Q(d, c)$$

Very efficient

Not very expressive

DPLL

Industry

First-Order

$$\forall x \exists y \quad Q(x, y) \lor \neg Q(y, f(x))$$
$$P(a) \lor Q(d, c)$$

Very expressive

Ground: not as efficient

Resolution/Superposition

Academia $\rightarrow$ Industry

From Ground to First-Order: Efficient at ground + Expressive?

Traditional Methods: Resolution

Reasoning Problem

Given a set of first order clauses S, prove S is unsatisfiable.

Resolution :

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

Example :

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

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

Traditional Methods: Resolution

Reasoning Problem

Given a set of first order clauses S, prove S is unsatisfiable.

Resolution :

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

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

Example :

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

Weaknesses:

- Inefficient in propositional case
- 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). For a quantifier free formula $\varphi(\bar{x})$;

$\forall\bar{x}\varphi(\bar{x})$ is unsatisfiable iff $\bigwedge_i \varphi(\bar{t}_i)$ is unsatisfiable,

for some ground terms $\bar{t}_1, \ldots, \bar{t}_n$.

Basic idea: Interleave instantiation with propositional reasoning.

Main issues:

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

Different approaches

Gilmore (1960): generation of ground instances

Robinson (1965): resolution

Plaisted et al (1992): hyper-linking

Plaisted & Zhu (2000): semantics-based instance generation

Letz & Stenz (2000): disconnection tableaux-type calculus

Hooker et al (2002): generation of instances with sem. selection

Baumgartner & Tinelli (2003): ME: Lifting of DPLL

Ganzinger & Korovin (2003): Inst-Gen calculus, modular ground reasoning

Claessen (2005): Equinox

... many instantiation based methods for different fragments/logics

Overview of the Inst-Gen procedure

First-Order Clauses

S

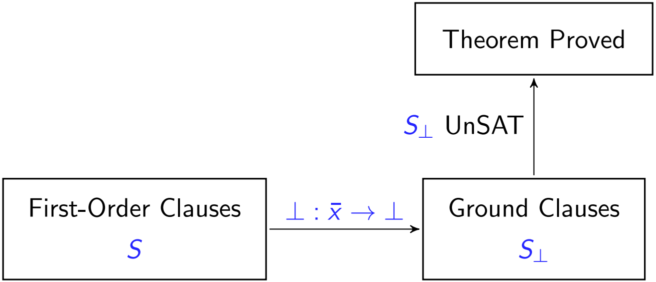
Theorem.[Ganzinger, Korovin LICS'03] Inst-Gen is sound and complete.

Overview of the Inst-Gen procedure

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

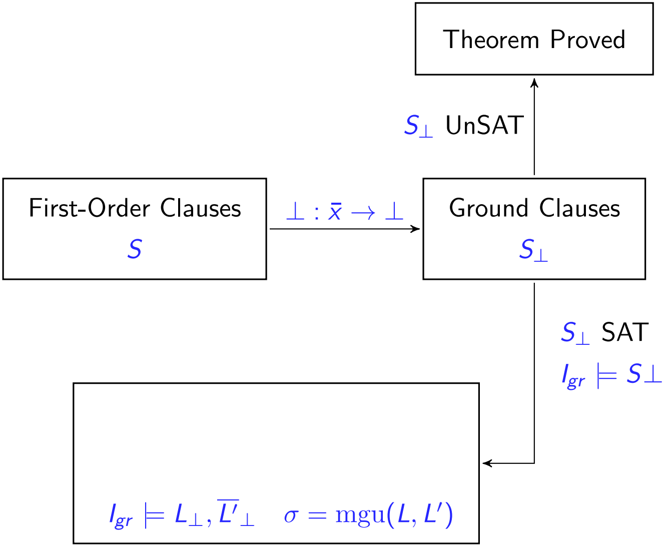
Theorem.[Ganzinger, Korovin LICS'03] Inst-Gen is sound and complete.

Overview of the Inst-Gen procedure



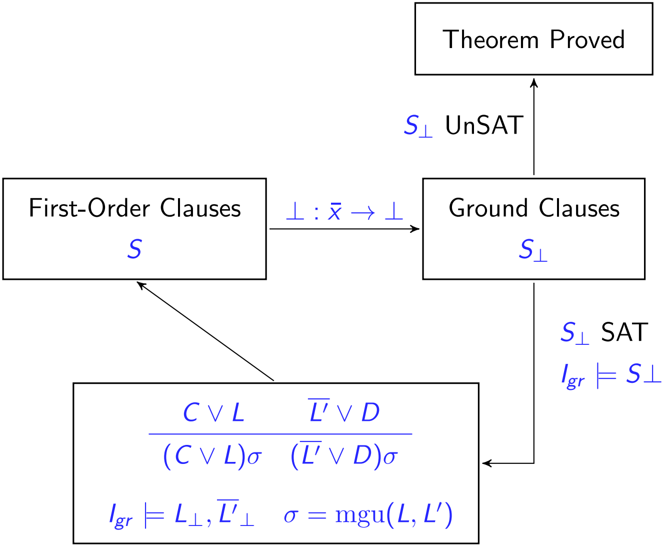
Theorem.[Ganzinger, Korovin LICS'03] Inst-Gen is sound and complete.

Overview of the Inst-Gen procedure



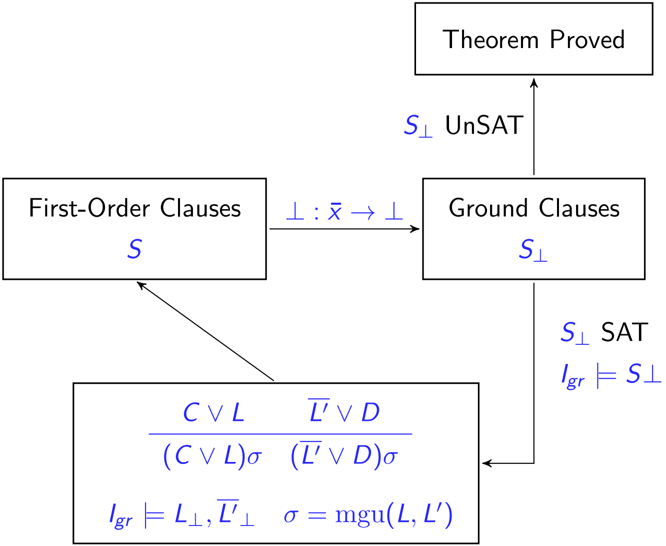
Theorem.[Ganzinger, Korovin LICS'03] Inst-Gen is sound and complete.

Overview of the Inst-Gen procedure



Theorem.[Ganzinger, Korovin LICS'03] Inst-Gen is sound and complete.

Overview of the Inst-Gen procedure



Theorem.[Ganzinger, Korovin LICS'03] Inst-Gen is sound and complete.

Example:

$$\boxed{\begin{array}{c} p(f(x), b) \vee q(x, y) \\[1em] \neg p(f(f(x)), y) \\[1em] \neg q(f(x), x) \end{array}}$$

Example:

$$\boxed{\begin{array}{c} p(f(x), b) \vee q(x, y) \\[1em] \neg p(f(f(x)), y) \\[1em] \neg q(f(x), x) \end{array}}
\qquad
\boxed{\begin{array}{c} p(f(\bot), b) \vee q(\bot, \bot) \\[1em] \neg p(f(f(\bot)), \bot) \\[1em] \neg q(f(\bot), \bot) \end{array}}$$

Example:

$$\boxed{\begin{array}{c} p(f(x), b) \lor q(x, y) \\[1em] \neg p(f(f(x)), y) \\[1em] \neg q(f(x), x) \end{array}} \qquad \boxed{\begin{array}{c} p(f(\bot), b) \lor q(\bot, \bot) \\[1em] \neg p(f(f(\bot)), \bot) \\[1em] \neg q(f(\bot), \bot) \end{array}}$$

Example:

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

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

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

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

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

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

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

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

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

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

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

Example:

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

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

Example:

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

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

The final set is propositionally unsatisfiable.

Resolution vs Inst-Gen

Resolution :

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

Instantiation :

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

Weaknesses of resolution:

Inefficient in the ground/EPR case

Length of clauses can grow fast

Recombination of clauses

No explicit model representation

Strengths of instantiation:

Modular ground reasoning

Length of clauses is fixed

Decision procedure for EPR

No recombination

Semantic selection

Redundancy elimination

Effective model representation

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 [Ganzinger, Korovin]. Redundant clauses/closures can be eliminated.

Consequences:

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

Simplifications by SAT/SMT solver *[Korovin IJCAR'08]*

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

Simplifications by SAT/SMT solver *[Korovin 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 ?

Simplifications by SAT/SMT solver [Korovin 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 ?

Basic idea:

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

Simplifications by SAT/SMT solver [Korovin 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 [Korovin IJCAR'08]

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 [Korovin IJCAR'08]

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

Global Ground Subsumption [Korovin IJCAR'08]

$$\frac{S_{gr}}{\begin{array}{l}\neg Q(a,b) \vee P(a) \vee P(b) \\ P(a) \vee Q(a,b) \\ \neg P(b)\end{array}} \qquad \frac{C}{P(a) \vee \cancel{Q(c,d)} \vee \cancel{Q(a,c)}}$$

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

Global Ground Subsumption [Korovin IJCAR'08]

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 *[Korovin IJCAR'08]*

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

Can we do more?

Non-Ground Simplifications by SAT/SMT *[Korovin IJCAR'08]*

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

Can we do more? Yes!

Ground solver can be used to simplify non-ground clauses.

Non-Ground Simplifications by SAT/SMT *[Korovin IJCAR'08]*

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

Can we do more? Yes!

Ground solver can be used to simplify non-ground clauses.

The main idea:

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

Non-Ground Simplifications by SAT/SMT *[Korovin IJCAR'08]*

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

Can we do more? Yes!

Ground solver can be used to simplify non-ground clauses.

The main idea:

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

Non-Ground Simplifications by SAT/SMT [Korovin IJCAR'08]

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

Can we do more? Yes!

Ground solver can be used to simplify non-ground clauses.

The main idea:

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

Non-Ground Simplifications by SAT/SMT *[Korovin IJCAR'08]*

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

Can we do more? Yes!

Ground solver can be used to simplify non-ground clauses.

The main idea:

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

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

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

Non-Ground Global Subsumption

Non-Ground Global Subsumption

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

Simplify first-order by purely ground reasoning!

Non-Ground Global Subsumption

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

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

Simplify first-order by purely ground reasoning!

Non-Ground Global Subsumption

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

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

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

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

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

Simplify first-order by purely ground reasoning!

Finer-grained control: closure orderings

Finer-grained control: replace ground clauses with ground closures.

Closure, a closure is a pair $C \cdot \sigma$,

where C is a clause and σ a grounding substitution

$$(A(a) \vee B(x)) \cdot [b/x]$$

Represents: ground clause $C\sigma$

$$A(a) \vee B(b)$$

Closure ordering: any total, well-founded ordering such that

$C\theta \cdot \tau \prec C \cdot \sigma$ if

- $C\sigma = C\theta\tau$, and
- θ properly instantiates C

Slogan: more specific representations take priority over less specific ones

Ex: $(p(a) \vee q(z)) \cdot [b/z] \prec (p(y) \vee q(z)) \cdot [a/y, b/z]$

Finer-grained control: closure orderings

Finer-grained control: replace ground clauses with ground closures.

Closure, a closure is a pair $C \cdot \sigma$,

where C is a clause and σ a grounding substitution

$$(A(a) \vee B(x)) \cdot [b/x]$$

Represents: ground clause $C\sigma$

$$A(a) \vee B(b)$$

Closure ordering: any total, well-founded ordering such that

$C\theta \cdot \tau \prec C \cdot \sigma$ if

- $C\sigma = C\theta\tau$, and
- θ properly instantiates C

Slogan: more specific representations take priority over less specific ones

Ex: $(p(a) \vee q(z)) \cdot [b/z] \prec (p(y) \vee q(z)) \cdot [a/y, b/z]$

Finer-grained control: closure orderings

Finer-grained control: replace ground clauses with ground closures.

Closure, a closure is a pair $C \cdot \sigma$,

where C is a clause and σ a grounding substitution

$$(A(a) \vee B(x)) \cdot [b/x]$$

Represents: ground clause $C\sigma$

$$A(a) \vee B(b)$$

Closure ordering: any total, well-founded ordering such that

$C\theta \cdot \tau \prec C \cdot \sigma$ if

- $C\sigma = C\theta\tau$, and
- θ properly instantiates C

Slogan: more specific representations take priority over less specific ones

Ex: $(p(a) \vee q(z)) \cdot [b/z] \prec (p(y) \vee q(z)) \cdot [a/y, b/z]$

Closure-based redundancy elimination

Definition call $C \cdot \sigma$ redundant in S if

- $C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n \models C \cdot \sigma$ and
- $C_1 \cdot \sigma_1, \ldots, C_n \cdot \sigma_n \prec C \cdot \sigma$

Theorem. [Ganzinger, Korovin]

Redundant closures (and clauses) can be eliminated.

Consequences:

- generalises usual redundancy
- new instantiation specific redundancies
  - blocking non-proper instances (merging variables) can be eliminated
  - dismatching constraints
- redundancy for inferences

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example:

$$\begin{array}{rr} p(x) \vee \underline{\neg q(f(x))} & (1) \\ \underline{p(f(x))} \vee \neg q(f(f(x))) & (2) \\ \underline{\mathrm{q(f(f(a)))}} & (3) \end{array}$$

Then the inference between (1) and (3) is redundant!

Why? the conclusion is represented twice $p(f(a)) \vee \neg q(f(f(a)))$

$p(f(x)) \vee \neg q(f(f(x))) \cdot [a/x] \prec p(x) \vee \neg q(f(x)) \cdot [f(a)/x]$

This can be represented as a dismatching constraint.

$$p(x) \vee \underline{\neg q(f(x))} \mid x \lhd_{ds} f(x)$$

How to make closures redundant? Instantiate!

Every proper instantiation inference makes closures redundant in the premise.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example:

$$p(x) \vee \underline{\neg q(f(x))} \qquad (1)$$
$$\underline{p(f(x))} \vee \neg q(f(f(x))) \quad (2)$$
$$\underline{\mathrm{q(f(f(a)))}} \qquad (3)$$

Then the inference between (1) and (3) is redundant!

Why? the conclusion is represented twice $p(f(a)) \vee \neg q(f(f(a)))$

$p(f(x)) \vee \neg q(f(f(x))) \cdot [a/x] \prec p(x) \vee \neg q(f(x)) \cdot [f(a)/x]$

This can be represented as a dismatching constraint.

$$p(x) \vee \underline{\neg q(f(x))} \mid x \triangleleft_{ds} f(x)$$

How to make closures redundant? Instantiate!

Every proper instantiation inference makes closures redundant in the premise.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example:

$$
\begin{array}{cr}
p(x) \vee \underline{\neg q(f(x))} & (1) \\
\underline{p(f(x))} \vee \neg q(f(f(x))) & (2) \\
\underline{\mathrm{q(f(f(a)))}} & (3)
\end{array}
$$

Then the inference between (1) and (3) is redundant!

Why? the conclusion is represented twice $p(f(a)) \vee \neg q(f(f(a)))$

$p(f(x)) \vee \neg q(f(f(x))) \cdot [a/x] \prec p(x) \vee \neg q(f(x)) \cdot [f(a)/x]$

This can be represented as a dismatching constraint.

$$p(x) \vee \underline{\neg q(f(x))} \mid x \lhd_{ds} f(x)$$

How to make closures redundant? Instantiate!

Every proper instantiation inference makes closures redundant in the premise.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example:

$$p(x) \vee \underline{\neg q(f(x))} \qquad (1)$$
$$\underline{p(f(x))} \vee \neg q(f(f(x))) \qquad (2)$$
$$\underline{\mathrm{q(f(f(a)))}} \qquad (3)$$

Then the inference between (1) and (3) is redundant!

Why? the conclusion is represented twice $p(f(a)) \vee \neg q(f(f(a)))$

$p(f(x)) \vee \neg q(f(f(x))) \cdot [a/x] \prec p(x) \vee \neg q(f(x)) \cdot [f(a)/x]$

This can be represented as a dismatching constraint.

$$p(x) \vee \underline{\neg q(f(x))} \mid x \lhd_{ds} f(x)$$

How to make closures redundant? Instantiate!

Every proper instantiation inference makes closures redundant in the premise.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example:

$$\begin{array}{cl} p(x) \vee \underline{\neg q(f(x))} & (1) \\ \underline{p(f(x))} \vee \neg q(f(f(x))) & (2) \\ \underline{\mathrm{q(f(f(a)))}} & (3) \end{array}$$

Then the inference between (1) and (3) is redundant!

Why? the conclusion is represented twice $p(f(a)) \vee \neg q(f(f(a)))$

$p(f(x)) \vee \neg q(f(f(x))) \cdot [a/x] \prec p(x) \vee \neg q(f(x)) \cdot [f(a)/x]$

This can be represented as a dismatching constraint.

$$p(x) \vee \underline{\neg q(f(x))} \mid x \lhd_{ds} f(x)$$

How to make closures redundant? Instantiate!

Every proper instantiation inference makes closures redundant in the premise.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example:

$$\begin{array}{rr} p(x) \vee \underline{\neg q(f(x))} & (1) \\ \underline{p(f(x))} \vee \neg q(f(f(x))) & (2) \\ \underline{\mathrm{q(f(f(a)))}} & (3) \end{array}$$

Then the inference between (1) and (3) is redundant!

Why? the conclusion is represented twice $p(f(a)) \vee \neg q(f(f(a)))$

$p(f(x)) \vee \neg q(f(f(x))) \cdot [a/x] \prec p(x) \vee \neg q(f(x)) \cdot [f(a)/x]$

This can be represented as a dismatching constraint.

$$p(x) \vee \underline{\neg q(f(x))} \mid x \lhd_{ds} f(x)$$

How to make closures redundant? Instantiate!

Every proper instantiation inference makes closures redundant in the premise.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example:

$$\begin{array}{cr} p(x) \vee \underline{\neg q(f(x))} & (1) \\ \underline{p(f(x))} \vee \neg q(f(f(x))) & (2) \\ \underline{\mathrm{q(f(f(a)))}} & (3) \end{array}$$

Then the inference between (1) and (3) is redundant!

Why? the conclusion is represented twice $p(f(a)) \vee \neg q(f(f(a)))$

$p(f(x)) \vee \neg q(f(f(x))) \cdot [a/x] \prec p(x) \vee \neg q(f(x)) \cdot [f(a)/x]$

This can be represented as a dismatching constraint.

$$p(x) \vee \underline{\neg q(f(x))} \mid x \lhd_{ds} f(x)$$

How to make closures redundant? Instantiate!

Every proper instantiation inference makes closures redundant in the premise.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example

$$
\begin{array}{ll}
A(f(y)) \vee D_1 & \neg A(x) \vee C \\
A(f^3(y)) \vee D_2 & \\
A(f^5(y)) \vee D_3 & \\
\ldots & \\
A(f^{i_n}(y)) \vee D_n &
\end{array}
$$

All other inferences with $\neg A(x) \vee C$ are blocked!

Premises inherit the constraints during instantiation inferences.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example

$$
\begin{array}{ll}
A(f(y)) \vee D_1 & \neg A(x) \vee C \mid x \lhd_{ds} f(y) \\
A(f^3(y)) \vee D_2 & \neg A(f(y)) \vee C \\
A(f^5(y)) \vee D_3 & \\
\ldots & \\
A(f^{i_n}(y)) \vee D_n &
\end{array}
$$

All other inferences with $\neg A(x) \vee C$ are blocked!

Premises inherit the constraints during instantiation inferences.

Dismatching Constraints *[Korovin (IJCAR'08, vol. HG'13)]*

Example

$$
\begin{array}{ll}
A(f(y)) \lor D_1 & \neg A(x) \lor C \mid x \lhd_{ds} f(y) \\
A(f^3(y)) \lor D_2 & \neg A(f(y)) \lor C \\
A(f^5(y)) \lor D_3 & \\
\dots & \\
A(f^{i_n}(y)) \lor D_n &
\end{array}
$$

All other inferences with $\neg A(x) \lor C$ are blocked!

Premises inherit the constraints during instantiation inferences.

Summary

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

- Inst-Gen is sound and complete for first-order logic
- combines efficient ground reasoning with first-order reasoning
- decision procedure for effectively propositional logic (EPR)
- redundancy elimination
  - usual: tautology elimination, strict subsumption
  - global subsumption:
    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies

Summary

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

- Inst-Gen is sound and complete for first-order logic
- combines efficient ground reasoning with first-order reasoning
- decision procedure for effectively propositional logic (EPR)
- redundancy elimination
  - usual: tautology elimination, strict subsumption
  - global subsumption:
    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies

Summary

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

- Inst-Gen is sound and complete for first-order logic
- combines efficient ground reasoning with first-order reasoning
- decision procedure for effectively propositional logic (EPR)
- redundancy elimination
  - usual: tautology elimination, strict subsumption
  - global subsumption:
    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies

Summary

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

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

    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies:
    - blocking non-proper instantiators
    - dismatching constraints

Summary

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

- Inst-Gen is sound and complete for first-order logic
- combines efficient ground reasoning with first-order reasoning
- decision procedure for effectively propositional logic (EPR)
- redundancy elimination
  - usual: tautology elimination, strict subsumption
  - global subsumption:
    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies:
    - blocking non-proper instantiators
    - dismatching constraints

Summary

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

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

    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies:
    - blocking non-proper instantiators
    - dismatching constraints

Summary

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

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

    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies:
    - blocking non-proper instantiators
    - dismatching constraints

Summary

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

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

    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies:
    - blocking non-proper instantiators
    - dismatching constraints

Summary

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

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

    non-ground simplifications using SAT/SMT reasoning
  - closure-based redundancies:
    - blocking non-proper instantiators
    - dismatching constraints

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

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

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

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

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

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

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:

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

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

Inst-Gen-Eq instantiation-based equational reasoning

f.-o. clauses
S

Theorem.[Ganzinger, Korovin CSL'04] Inst-Gen-Eq is sound and complete.

Inst-Gen-Eq instantiation-based equational reasoning

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

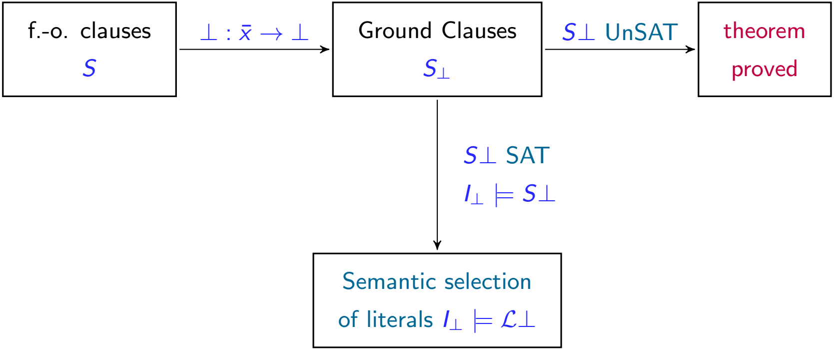
Theorem.[Ganzinger, Korovin CSL'04] Inst-Gen-Eq is sound and complete.

Inst-Gen-Eq instantiation-based equational reasoning

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

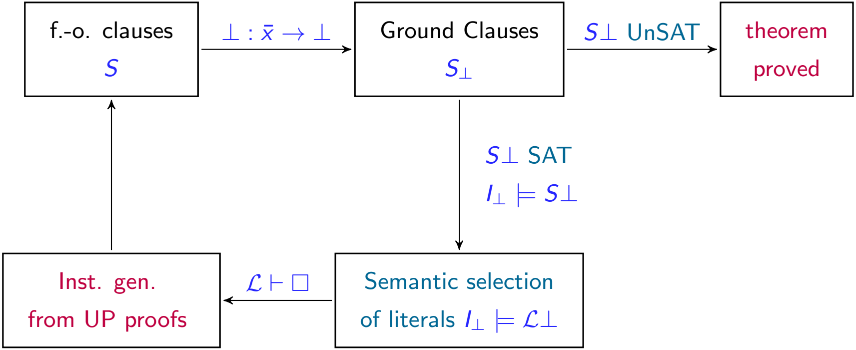
Theorem.[Ganzinger, Korovin CSL'04] Inst-Gen-Eq is sound and complete.

Inst-Gen-Eq instantiation-based equational reasoning



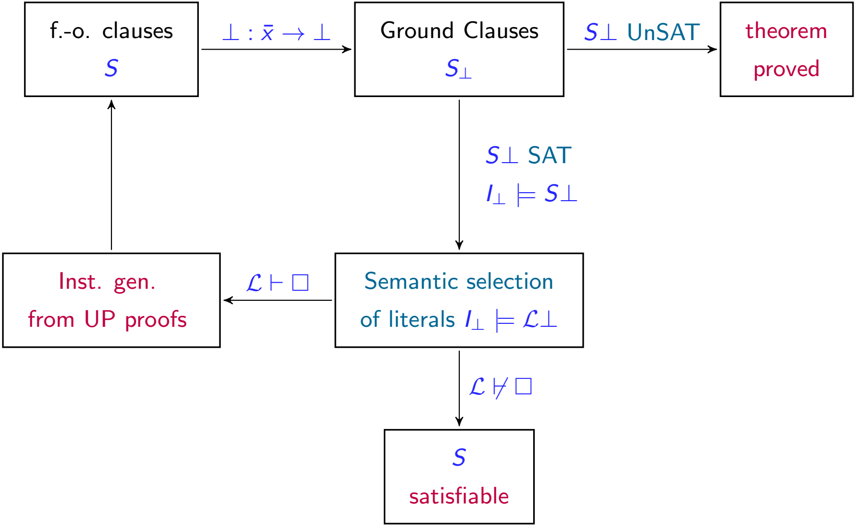
Theorem.[Ganzinger, Korovin CSL'04] Inst-Gen-Eq is sound and complete.

Inst-Gen-Eq instantiation-based equational reasoning



Theorem.[Ganzinger, Korovin CSL'04] Inst-Gen-Eq is sound and complete.

Inst-Gen-Eq instantiation-based equational reasoning



Theorem.[Ganzinger, Korovin CSL'04] Inst-Gen-Eq is sound and complete.

Inst-Gen-Eq: Key properties

Inst-Gen-Eq is

- sound and complete for first-order logic with equality
- combines SMT for ground reasoning and superposition-based unit reasoning
- 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

New technical issue: Potentially we need to consider all unit-superposition proofs!

Inst-Gen-Eq: Key properties

Inst-Gen-Eq is

- sound and complete for first-order logic with equality
- combines SMT for ground reasoning and superposition-based unit reasoning
- 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

New technical issue: Potentially we need to consider all unit-superposition proofs!

Inst-Gen-Eq: Key properties

Inst-Gen-Eq is

- sound and complete for first-order logic with equality
- combines SMT for ground reasoning and superposition-based unit reasoning
- 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

New technical issue: Potentially we need to consider all unit-superposition proofs!

Inst-Gen-Eq: Key properties

Inst-Gen-Eq is

- sound and complete for first-order logic with equality
- combines SMT for ground reasoning and superposition-based unit reasoning
- 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

New technical issue: Potentially we need to consider all unit-superposition proofs!

Inst-Gen-Eq: Key properties

Inst-Gen-Eq is

- sound and complete for first-order logic with equality
- combines SMT for ground reasoning and superposition-based unit reasoning
- 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

New technical issue: Potentially we need to consider all unit-superposition proofs!

Inst-Gen-Eq: Key properties

Inst-Gen-Eq is

- sound and complete for first-order logic with equality
- combines SMT for ground reasoning and superposition-based unit reasoning
- 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

New technical issue: Potentially we need to consider all unit-superposition proofs!

Inst-Gen-Eq: Key properties

Inst-Gen-Eq is

- sound and complete for first-order logic with equality
- combines SMT for ground reasoning and superposition-based unit reasoning
- 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

New technical issue: Potentially we need to consider all unit-superposition proofs!

Labelled Unit Superposition *[Korovin, Sticksel LPAR'10]*

General idea: Dismatching constraints can be used to block already derived proofs!

Unit superposition with dismatching constraints:

$$\frac{(l \simeq r) \mid [\ D_1\ ] \quad L[l'] \mid [\ D_2\ ]}{L[r]\theta \mid [\ (D_1 \wedge D_2)\theta\ ]}\ (\theta) \qquad\qquad \frac{s \not\simeq t \mid [\ D\ ]}{\square}\ (\mu)$$

where (i) $\theta = \mathrm{mgu}(l, l')$; (ii) l' is not a variable; (iii) for some grounding substitution σ, satisfying $(D_1 \wedge D_2)\theta$, $l\sigma \succ r\sigma$; (iv) $\mu = \mathrm{mgu}(s, t)$; (v) $D\mu$ is satisfiable.

Next technical issue: The same unit literal can

- correspond to different clauses,
- have different dismatching constraints
- be represented many times in the same proof search

Solution: labelled approach

Labelled Unit Superposition [Korovin, Sticksel LPAR'10]

General idea: Dismatching constraints can be used to block already derived proofs!

Unit superposition with dismatching constraints:

$$\frac{(l \simeq r) \mid [\ D_1\ ] \quad L[l'] \mid [\ D_2\ ]}{L[r]\theta \mid [\ (D_1 \wedge D_2)\theta\ ]}\ (\theta) \qquad\qquad \frac{s \not\simeq t \mid [\ D\ ]}{\square}\ (\mu)$$

where (i) $\theta = \mathrm{mgu}(l, l')$; (ii) l' is not a variable; (iii) for some grounding substitution σ, satisfying $(D_1 \wedge D_2)\theta$, $l\sigma \succ r\sigma$; (iv) $\mu = \mathrm{mgu}(s, t)$; (v) $D\mu$ is satisfiable.

Next technical issue: The same unit literal can

- correspond to different clauses,
- have different dismatching constraints
- be represented many times in the same proof search

Solution: labelled approach

Labelled Unit Superposition *[Korovin, Sticksel LPAR'10]*

General idea: Dismatching constraints can be used to block already derived proofs!

Unit superposition with dismatching constraints:

$$\frac{(l \simeq r) \mid [\ D_1\ ] \quad L[l'] \mid [\ D_2\ ]}{L[r]\theta \mid [\ (D_1 \wedge D_2)\theta\ ]}\ (\theta) \qquad\qquad \frac{s \not\simeq t \mid [\ D\ ]}{\square}\ (\mu)$$

where (i) $\theta = \mathrm{mgu}(l, l')$; (ii) l' is not a variable; (iii) for some grounding substitution σ, satisfying $(D_1 \wedge D_2)\theta$, $l\sigma \succ r\sigma$; (iv) $\mu = \mathrm{mgu}(s, t)$; (v) $D\mu$ is satisfiable.

Next technical issue: The same unit literal can

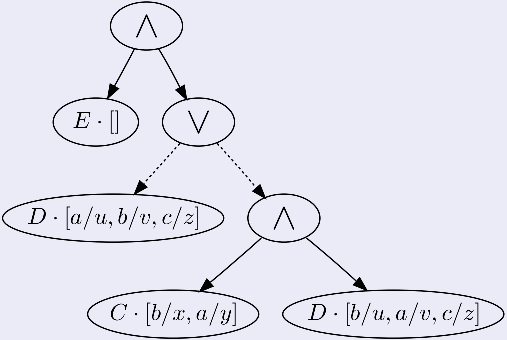
- correspond to different clauses,
- have different dismatching constraints
- be represented many times in the same proof search

Solution: labelled approach

Tree Labelled Unit Superposition

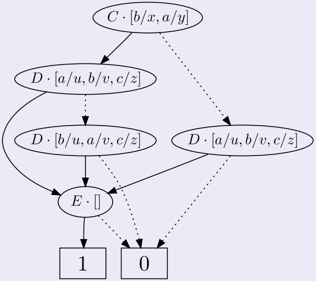
- Preserve Boolean structure of proofs
- Closure is a propositional variable in an AND/OR tree
- Conjunction $\wedge$ in superposition, disjunction $\vee$ in merging

Label of the Contradiction $\Box$



OBDD Labelled Unit Superposition

Label of the contradiction $\Box$



Disadvantages of trees

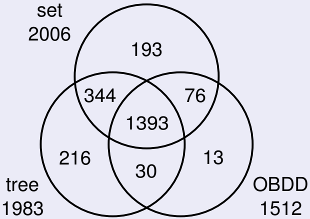
- Not produced in normal form
- Sequence of inferences determines shape
- Potential growth *ad infinitum*

- OBDD as normal form
- Maintenance effort
- Reordering required

Labels: Sets vs. Trees vs. OBDDs

iProver-Eq – CVC3 as a background solver on pure equational problems.
(developed with Christoph Sticksel)

Solved equational problems



Features

	Normal form	Precise elim.
Sets	yes	no
Trees	no	yes
OBDDs	yes	yes

[Korovin, Sticksel LPAR'10]

Theory instantiation

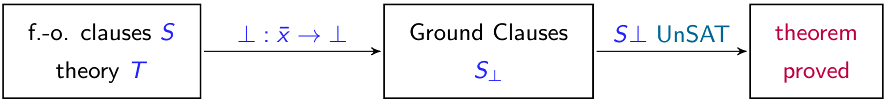
Theory instantiation *[Ganzinger, Korovin LPAR'06]*

f.-o. clauses S
theory T

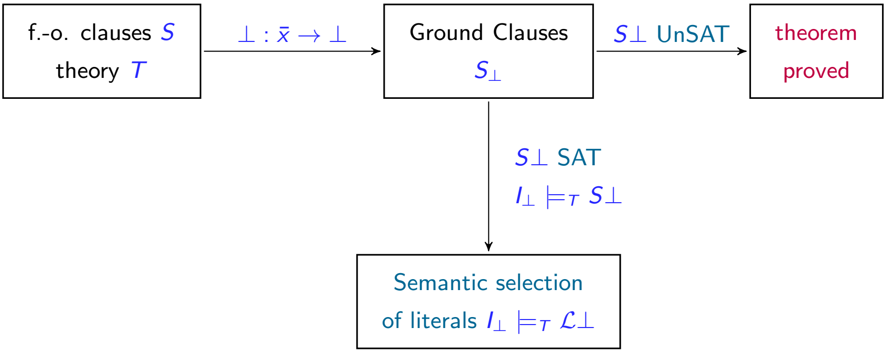
Theory instantiation [Ganzinger, Korovin LPAR'06]

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

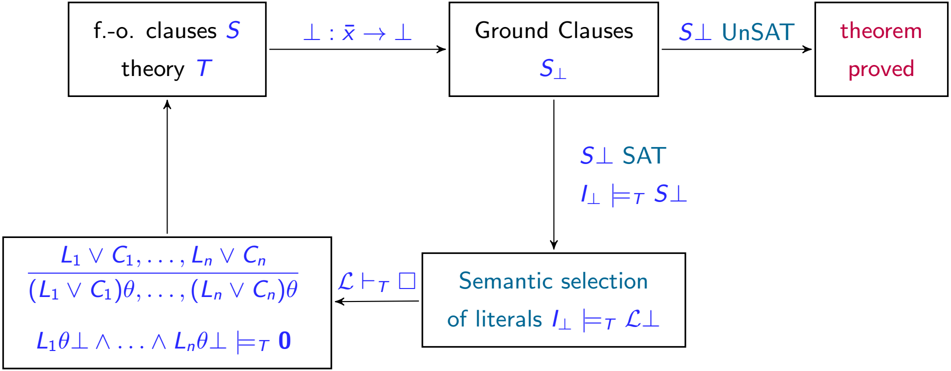
Theory instantiation *[Ganzinger, Korovin LPAR'06]*



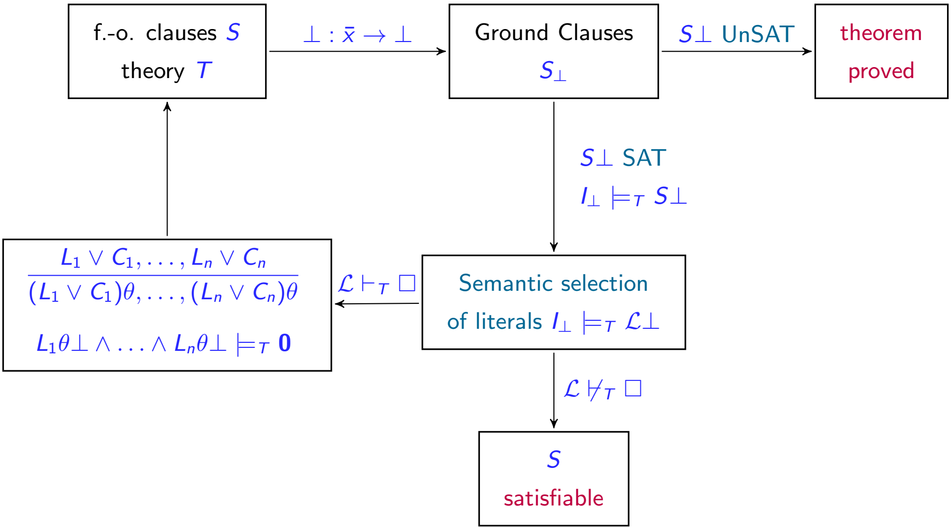
Theory instantiation [Ganzinger, Korovin LPAR'06]



Theory instantiation *[Ganzinger, Korovin LPAR'06]*



Theory instantiation [Ganzinger, Korovin LPAR'06]



Theory instantiation

Conditions on completeness:

- complete ground reasoning modulo T
- answer completeness of unit reasoning modulo T
- T is universal

Answer completeness: If $L_1\tau \wedge \ldots \wedge L_n\tau \models_T \Box$ for ground τ. Then

$$\frac{L_1, \ldots, L_n}{L_1\theta, \ldots, L_n\theta}\ UC$$

such that θ is a generalization of τ and $L_1\theta\bot, \ldots, L_n\theta\bot \vdash_T \Box$

Theorem. Theory instantiation is sound and complete under these conditions.

Theory instantiation

Conditions on completeness:

- complete ground reasoning modulo T
- answer completeness of unit reasoning modulo T
- T is universal

Answer completeness: If $L_1\tau \wedge \ldots \wedge L_n\tau \models_T \Box$ for ground τ. Then

$$\frac{L_1, \ldots, L_n}{L_1\theta, \ldots, L_n\theta}\ UC$$

such that θ is a generalization of τ and $L_1\theta\bot, \ldots, L_n\theta\bot \vdash_T \Box$

Theorem. Theory instantiation is sound and complete under these conditions.

Theory instantiation

Conditions on completeness:

- complete ground reasoning modulo T
- answer completeness of unit reasoning modulo T
- T is universal

Answer completeness: If $L_1\tau \wedge \ldots \wedge L_n\tau \models_T \Box$ for ground τ. Then

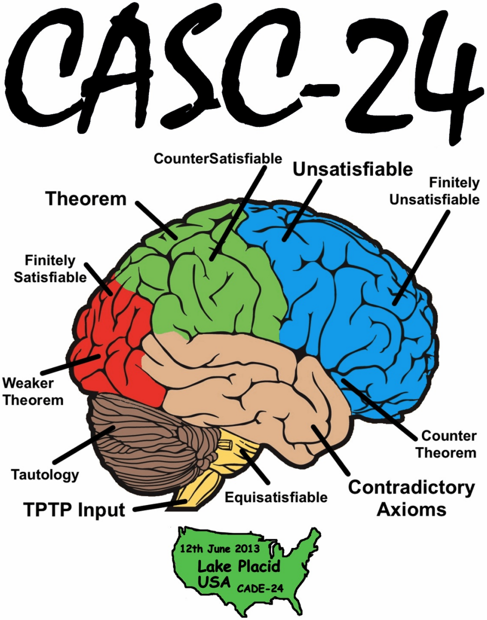
$$\frac{L_1, \ldots, L_n}{L_1\theta, \ldots, L_n\theta}\ UC$$

such that θ is a generalization of τ and $L_1\theta\bot, \ldots, L_n\theta\bot \vdash_T \Box$

Theorem. Theory instantiation is sound and complete under these conditions.

Evaluation

CASC 2013


CASC-24
CounterSatisfiable
Unsatisfiable
Finitely Unsatisfiable
Theorem
Finitely Satisfiable
Weaker Theorem
Tautology
TPTP Input
Equisatisfiable
Counter Theorem
Contradictory Axioms
12th June 2013
Lake Placid
USA
CADE-24

CASC 2013 results

General first-order (FOF) 300 problems

	Vampire	E	iProver	E-KRHyper	Prover9
prob	281	249	167	122	119
time	12	29	12	8	12

Effectively propositional 100 problems

	iProver	Vampire	PEPR	E	EKRHyper
prob	81	47	43	23	8
time	27	15	26	50	27

First-order satisfiability (FNT) 150 problems

	iProver	Paradox	CVC4	E	Nitrox	Vampire
prob	122	99	96	79	79	78
time	52	2	25	20	29	30

Non-cyclic sorts for first-order satisfiability [Korovin FroCoS'13]

CASC 2013 results

General first-order (FOF) 300 problems

	Vampire	E	iProver	E-KRHyper	Prover9
prob	281	249	167	122	119
time	12	29	12	8	12

Effectively propositional 100 problems

	iProver	Vampire	PEPR	E	EKRHyper
prob	81	47	43	23	8
time	27	15	26	50	27

First-order satisfiability (FNT) 150 problems

	iProver	Paradox	CVC4	E	Nitrox	Vampire
prob	122	99	96	79	79	78
time	52	2	25	20	29	30

Non-cyclic sorts for first-order satisfiability [Korovin FroCoS'13]

CASC 2013 results

General first-order (FOF) 300 problems

	Vampire	E	iProver	E-KRHyper	Prover9
prob	281	249	167	122	119
time	12	29	12	8	12

Effectively propositional 100 problems

	iProver	Vampire	PEPR	E	EKRHyper
prob	81	47	43	23	8
time	27	15	26	50	27

First-order satisfiability (FNT) 150 problems

	iProver	Paradox	CVC4	E	Nitrox	Vampire
prob	122	99	96	79	79	78
time	52	2	25	20	29	30

Non-cyclic sorts for first-order satisfiability [Korovin FroCoS'13]

Effectively propositional logic (EPR)

Effectively Propositional Logic (EPR)

EPR: No functions except constants: $P(x, y) \vee \neg Q(c, y)$

Effectively Propositional Logic (EPR)

EPR: No functions except constants: $P(x, y) \vee \neg Q(c, y)$

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

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

Verification:

$$\forall A(\texttt{wren}_{h1} \wedge A = \texttt{wraddrFunc} \rightarrow \\ \forall B(\texttt{range}_{[35,0]}(B) \rightarrow (\texttt{imem}'(A, B) \leftrightarrow \texttt{iwrite}(B)))).$$

Applications:

- Hardware Verification (Intel)
- Planning/Scheduling
- Finite model reasoning

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

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:

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

General lemmas:

$$\neg a(x) \vee b(x) \qquad \neg b(x) \vee \mathtt{mem}(x, y)$$
$$a(x) \vee \mathtt{mem}(x, y)$$

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:

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

General lemmas:

$$\neg a(x) \vee b(x) \qquad \neg b(x) \vee \mathtt{mem}(x, y)$$
$$a(x) \vee \mathtt{mem}(x, y) \qquad \mathtt{mem}(x, y)$$

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:

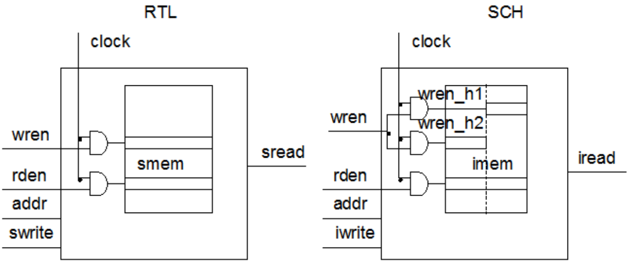
$$\neg\mathtt{mem}(a_1, x_1) \vee \neg\mathtt{mem}(a_2, x_2) \vee \ldots \neg\mathtt{mem}(a_n, x_n)$$
$$\mathtt{mem}(b_1, x_1) \vee \mathtt{mem}(b_2, x_2) \vee \ldots \vee \mathtt{mem}(b_n, x_n)$$

General lemmas:

$$\neg a(x) \vee b(x) \qquad \cancel{\neg b(x) \vee \mathtt{mem}(x, y)}$$
$$\cancel{a(x) \vee \mathtt{mem}(x, y)} \qquad \mathtt{mem}(x, y)$$

More expressive logics can speed up calculations!

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) used at Intel, IBM

EPR-based BMC Navarro-Perez, Voronkov (CADE'07)

EPR encoding:

- $s_0, \ldots, s_k$ constants denote unrolling bounds
- first-order formulas $I(S)$, $P(S)$, $T(S, S')$
- next state predicate $Next(S, S')$

BMC can be encoded

$I(s_0); \neg P(s_k);$	initial and final states
$\forall S, S'(Next(S, S') \rightarrow T(S, S'));$	transition relation
$Next(s_0, s_1); Next(s_1, s_2); \ldots Next(s_{k-1}, s_k);$	next state relation

- EPR encoding provides succinct representation
- avoids copying transition relation
- reasoning can be done at higher level

BMC with bit-vectors, memories:

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

EPR-based BMC *Navarro-Perez, Voronkov (CADE'07)*

EPR encoding:

- $s_0, \ldots, s_k$ constants denote unrolling bounds
- first-order formulas $I(S)$, $P(S)$, $T(S, S')$
- next state predicate $Next(S, S')$

BMC can be encoded

$I(s_0); \neg P(s_k);$	initial and final states
$\forall S, S'(Next(S, S') \rightarrow T(S, S'));$	transition relation
$Next(s_0, s_1); Next(s_1, s_2); \ldots Next(s_{k-1}, s_k);$	next state relation

- EPR encoding provides succinct representation
- avoids copying transition relation
- reasoning can be done at higher level

BMC with bit-vectors, memories:

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

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 outperforms highly optimised Intel SAT-based model checker.

Implementation

iProver general features

- Inst-Gen also uses SAT solver and resolution for simplifications
- Query answering: using answer substitutions
- Finite model finding: based on EPR/sort inference/non-cyclic sorts
- Bounded model checking mode: (Intel format)
- Proof representation: non-trivial due to SAT solver simplifications
- Model representation: using formulas in term algebra;
  special model representation for hardware BMC

iProver implementation features

iProver is implemented in OCaml, around 50,000 LOC

Core:

- Inst-Gen Given clause algorithm
- SAT solvers for ground reasoning: MiniSAT, PicoSAT, Lingeling
- strategy scheduling
- preprocessing
- splitting with naming

Simplifications:

- Literal selection
- Subsumption (forward/backward)
- Subsumption resolution (forward/backward)
- Dismatching constraints
- Blocking non-proper instantiators
- Global subsumption: SAT solver is used for non-ground simplifications

Inst-Gen given clause algorithm

Passive: clauses that are waiting to participate in inferences

- priority queues based on lexicographic combinations of parameters
  - $--$ *inst_pass_queue1* $[-conj\_dist; +conj\_symb; -num\_var]$
  - $--$ *inst_pass_queue2* $[+age; -num\_symb]$

Active: clauses between which all inferences are done

- unification index on selected literals

  Non-perfect discrimination trees

Given clause: C

1. C – next clause from the top of Passive
2. simplify C: compressed feature indexes
3. perform all inferences between C and Active
4. add all conclusions to passive
5. add $\perp$-grounding of conclusions to the SAT solver

Inst-Gen given clause algorithm

Passive: clauses that are waiting to participate in inferences

- priority queues based on lexicographic combinations of parameters
  - $--$ *inst_pass_queue1* $[-conj\_dist; +conj\_symb; -num\_var]$
  - $--$ *inst_pass_queue2* $[+age; -num\_symb]$

Active: clauses between which all inferences are done

- unification index on selected literals

  Non-perfect discrimination trees

Given clause: C

1. C – next clause from the top of Passive
2. simplify C: compressed feature indexes
3. perform all inferences between C and Active
4. add all conclusions to passive
5. add $\perp$-grounding of conclusions to the SAT solver

Inst-Gen given clause algorithm

Passive: clauses that are waiting to participate in inferences

- priority queues based on lexicographic combinations of parameters
  - $--$ *inst_pass_queue1* $[-conj\_dist; +conj\_symb; -num\_var]$
  - $--$ *inst_pass_queue2* $[+age; -num\_symb]$

Active: clauses between which all inferences are done

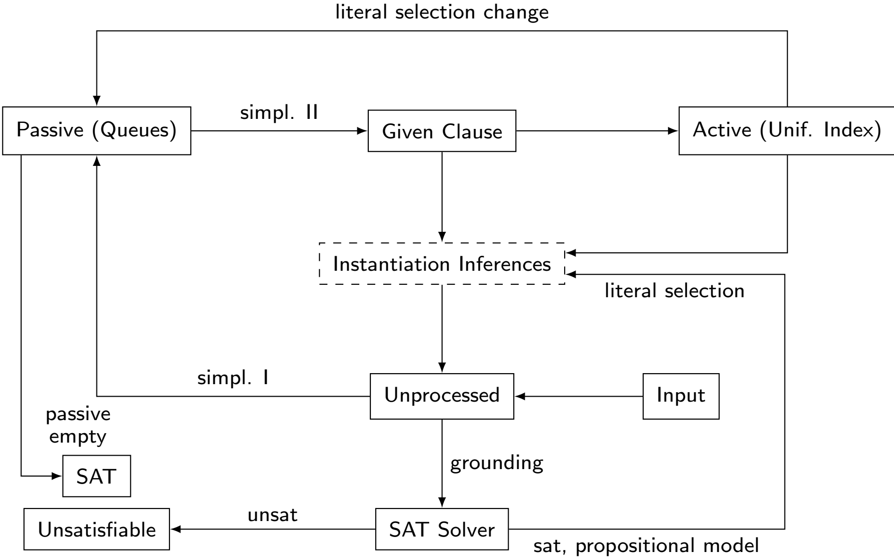
- unification index on selected literals

  Non-perfect discrimination trees

Given clause: C

1. C – next clause from the top of Passive
2. simplify C: compressed feature indexes
3. perform all inferences between C and Active
4. add all conclusions to passive
5. add $\perp$-grounding of conclusions to the SAT solver

Inst-Gen Loop



[Korovin (Essays in Memory of Harald Ganzinger 2013])

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.

Indexing

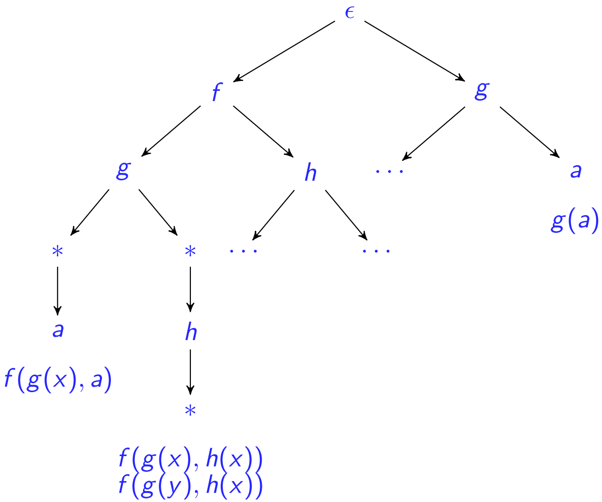
Why indexing:

- 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.

Discrimination trees



Efficient filtering unification, matching and generalisation candidates

Feature vector index

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz'04] 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

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz'04] 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

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz'04] 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

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz'04] 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
- $\dots$

Feature vector index

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz'04] 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

Subsumption is very expensive and usual indexing are complicated.

Feature vector index [Schulz'04] 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 $C = p(f(f(x))) \vee \neg p(g(y))$ is
$fv(C) = [2, 2, 1]$

Arrange feature vectors in a trie data structure.

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

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 $C = p(f(f(x))) \vee \neg p(g(y))$ is

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

Arrange feature vectors in a trie data structure.

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 *[Korovin (iProver'08)]*

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.

Compressed feature vector index *[Korovin (iProver'08)]*

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.

Compressed feature vector index *[Korovin (iProver'08)]*

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.

Summary

iProver is a theorem prover for full clausal first-order logic which features

- Query answering: using answer substitutions
- Finite model finding: based on EPR/sort inference/non-cyclic sorts
- Bounded model checking mode: (Intel format)
- Proof representation: non-trivial due to SAT solver simplifications
- Model representation: using formulas in term algebra;
  special model representation for hardware BMC

iProver has solid performance over the whole range of TPTP.

iProver excels on EPR problems and in turn on satisfiability, bounded model checking and other encodings into EPR.

PhD opportunities at the University of Manchester

PhD opportunities in reasoning, logic and verification, please contact:

korovin@cs.man.ac.uk