Automated reasoning for first-order logic
Theory, Practice and Challenges

Konstantin Korovin¹
The University of Manchester
UK

korovin@cs.man.ac.uk

Part II

¹supported 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.

<table>
<thead>
<tr>
<th>Ground (SAT/SMT)</th>
<th>First-Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(a) \lor Q(c, d)$</td>
<td>$\forall x \exists y \ Q(x, y) \lor \neg Q(y, f(x))$</td>
</tr>
<tr>
<td>$\neg P(a) \lor Q(d, c)$</td>
<td>$P(a) \lor Q(d, c)$</td>
</tr>
</tbody>
</table>

**Very efficient**

**Not very expressive**

**DPLL**

**Industry**

**Ground:** not as efficient

**Very expressive**

**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:

$$\frac{C \lor L \quad \overline{L}' \lor D}{(C \lor D)\sigma}$$

Example:

$$\frac{Q(x) \lor P(x) \quad \neg P(a) \lor R(y)}{Q(a) \lor R(y)}$$

Weaknesses:

- Inefficient in propositional case
- Length of clauses can grow fast
- Recombination of clauses
- No effective model representation
Traditional Methods: Resolution

Reasoning Problem

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

Resolution:

$$
\begin{align*}
    C \lor L & \quad \overline{L'} \lor D \\
    \hline
    (C \lor D)_{\sigma}
\end{align*}
$$

Example:

$$
\begin{align*}
    Q(x) \lor P(x) & \quad \neg P(a) \lor R(y) \\
    \hline
    Q(a) \lor R(y)
\end{align*}
$$

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


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


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

First-Order Clauses

\[ S \]

Ground Clauses

\[ S_\bot \]

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

First-Order Clauses

\( S \)

\( \bot : \bar{x} \rightarrow \bot \)

Ground Clauses

\( S_\bot \)

Theorem Proved

\( S_\bot \) UnSAT

Theorem.

[Ganzinger, Korovin LICS'03] Inst-Gen is sound and complete.
Overview of the Inst-Gen procedure

First-Order Clauses

$S$

$\perp$ : $\overline{x} \rightarrow \perp$

Ground Clauses

$S_{\perp}$

Theorem Proved

$S_{\perp}$ UnSAT

$S_{\perp}$ SAT

$l_{gr} \models S_{\perp}$

$l_{gr} \models L_{\perp}, L'_{\perp}$

$\sigma = mgu(L, L')$

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

First-Order Clauses

\[ S \]

\[ \bot : \bar{x} \rightarrow \bot \]

Ground Clauses

\[ S_{\bot} \]

Theorem Proved

\[ S_{\bot} \text{ UnSAT} \]

\[ S_{\bot} \text{ SAT} \]

\[ I_{gr} \models S_{\bot} \]

\[
\frac{C \lor L}{(C \lor L)\sigma} \quad \frac{\bar{L}' \lor D}{(\bar{L}' \lor D)\sigma}
\]

\[ I_{gr} \models L_{\bot}, \bar{L}'_{\bot} \quad \sigma = \text{mgu}(L, L') \]

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

First-Order Clauses

\( S \)

\( \bot : \overline{x} \rightarrow \bot \)

Ground Clauses

\( S_\bot \)

\( S_\bot \) UnSAT

\( S_\bot \) SAT

\( I_{gr} \models S_\bot \)

Theorem.

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

\[
p(f(x), b) \lor q(x, y) \\
\neg p(f(f(x)), y) \\
\neg q(f(x), x)
\]
Example:

\[ p(f(x), b) \lor q(x, y) \]
\[ \neg p(f(f(x)), y) \]
\[ \neg q(f(x), x) \]

\[ p(f(\bot), b) \lor q(\bot, \bot) \]
\[ \neg p(f(f(\bot)), \bot) \]
\[ \neg q(f(\bot), \bot) \]
Example:

\[ p(f(x), b) \lor q(x, y) \]

\[ \neg p(f(f(x)), y) \]

\[ \neg q(f(x), x) \]

\[ p(f(\bot), b) \lor q(\bot, \bot) \]

\[ \neg p(f(f(\bot)), \bot) \]

\[ \neg q(f(\bot), \bot) \]
Example:

\[ p(f(x), b) \lor q(x, y) \]
\[ \neg p(f(f(x)), y) \]
\[ \neg q(f(x), x) \]
\[ p(f(f(x)), b) \lor q(f(x), y) \]
\[ \neg p(f(f(x)), b) \]
\[ p(f(x), b) \lor q(x, y) \]
\[ \neg p(f(f(x)), y) \]
\[ \neg q(f(x), x) \]

\[ p(f(\bot), b) \lor q(\bot, \bot) \]
\[ \neg p(f(f(\bot)), \bot) \]
\[ \neg q(f(\bot), \bot) \]
Example:

\[
\begin{align*}
\text{LHS:} & \quad p(f(x), b) \lor q(x, y) \\
& \quad \neg p(f(f(x)), y) \\
& \quad \neg q(f(x), x)
\end{align*}
\]

\[
\begin{align*}
\text{RHS:} & \quad p(f(\bot), b) \lor q(\bot, \bot) \\
& \quad \neg p(f(f(\bot)), \bot) \\
& \quad \neg q(f(\bot), \bot)
\end{align*}
\]
Example:

\[ p(f(x), b) \lor q(x, y) \]
\[ \neg p(f(f(x)), y) \]
\[ \neg q(f(x), x) \]

\[ p(f(f(x)), b) \lor q(f(x), y) \]
\[ \neg p(f(f(x)), b) \]
\[ p(f(x), b) \lor q(x, y) \]
\[ \neg p(f(f(x)), y) \]
\[ \neg q(f(x), x) \]

\[ p(f(\bot), b) \lor q(\bot, \bot) \]
\[ \neg p(f(f(\bot)), \bot) \]
\[ \neg q(f(\bot), \bot) \]

\[ p(f(f(\bot)), b) \lor q(f(\bot), \bot) \]
\[ \neg p(f(f(\bot)), b) \]
\[ p(f(\bot), b) \lor q(\bot, \bot) \]
\[ \neg p(f(f(\bot)), \bot) \]
\[ \neg q(f(\bot), \bot) \]

The final set is propositionally \textbf{unsatisfiable}.
Resolution vs Inst-Gen

Resolution:

\[
\begin{align*}
(C \lor L) & \quad (\overline{L'} \lor D) \\
\hline
(C \lor D)\sigma & \\
\sigma &= \text{mgu}(L, L')
\end{align*}
\]

Instantiation:

\[
\begin{align*}
(C \lor L) & \quad (\overline{L'} \lor D) \\
\hline
(C \lor L)\sigma & \quad (\overline{L'} \lor D)\sigma \\
\sigma &= \text{mgu}(L, L')
\end{align*}
\]

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 presentation
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.

- $P(a) \models Q(b) \lor P(a)$
- $P(a) \prec Q(b) \lor \overline{P(a)}$
Redundancy Elimination

The key to efficiency is redundancy elimination.

Ground clause $C$ is redundant if

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

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

Abstract redundancy:

\[ C_1, \ldots, C_n \models C \]

\[ C_1, \ldots, C_n \preceq C \]

\[ S_{gr} \models C \] — ground solver

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:

\[ C_1, \ldots, C_n \models C \]
\[ C_1, \ldots, C_n \prec C \]

\[ S_{gr} \models C \] — ground solver follows from smaller ?

Basic idea:

- split \( D \subset C \)
- check \( S_{gr} \models D \)
- add \( D \) to \( S \) and remove \( C \)
Can off-the-shelf ground solver be used to simplify ground clauses?

Abstract redundancy:
\[ C_1, \ldots, C_n \models C \quad S_{gr} \models C \quad \text{ground solver follows from smaller?} \]

Basic idea:
- split \( D \subset C \)
- check \( S_{gr} \models D \)
- add \( D \) to \( S \) and remove \( C \)

Global ground subsumption:
\[
\frac{D \lor C'}{D}
\]
where \( S_{gr} \models D \) and \( C' \neq \emptyset \)
Global Ground Subsumption [Korovin IJCAR’08]

\[ S_{gr} \]

\[ \neg Q(a, b) \lor P(a) \lor P(b) \]
\[ P(a) \lor Q(a, b) \]
\[ \neg P(b) \]

\[ C \]

\[ P(a) \lor Q(c, d) \lor Q(a, c) \]
Global Ground Subsumption [Korovin IJCAR’08]

\[
\begin{align*}
S_{gr} & \\
\neg Q(a, b) \lor P(a) \lor P(b) & \\
P(a) \lor Q(a, b) & \\
\neg P(b) & \\
\end{align*}
\]

\[
\begin{align*}
C & \\
P(a) \lor Q(c, d) \lor Q(a, c) & \\
\end{align*}
\]
Global Ground Subsumption \cite{KorovinIJCAR08}

\[
\begin{align*}
S_{gr} & \\
\neg Q(a, b) & \lor P(a) \lor P(b) \\
P(a) & \lor Q(a, b) \\
\neg P(b)
\end{align*}
\]

\[
\begin{align*}
C & \\
P(a) & \lor Q(c, d) \lor Q(a, c)
\end{align*}
\]

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

\[ S_{gr} \]
\[ \neg Q(a, b) \lor P(a) \lor P(b) \]
\[ P(a) \lor Q(a, b) \]
\[ \neg P(b) \]

\[ C \]
\[ P(a) \lor Q(c, d) \lor Q(a, c) \]

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

Global Ground Subsumption 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?
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.
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:

\[ S_{gr} \models \forall \bar{x} \, C(\bar{x}) \]
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:

\[
S_{gr} \models \forall \bar{x} C(\bar{x}) \quad S_{gr} \models C(\bar{d}) \quad \text{for fresh} \quad \bar{d}
\]
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:

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

\[ C_1(\bar{x}), \ldots, C_n(\bar{x}) \in S \\
C_1(\bar{d}), \ldots, C_n(\bar{d}) \models C(\bar{d}) \]
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:

\[ S_{gr} \models \forall \bar{x} C(\bar{x}) \]
\[ C_1(\bar{x}), \ldots, C_n(\bar{x}) \in S \]
\[ C_1(\bar{x}), \ldots, C_n(\bar{x}) \prec C(\bar{x}) \]

\[ S_{gr} \models C(\bar{d}) \text{ for fresh } \bar{d} \]
\[ C_1(\bar{d}), \ldots, C_n(\bar{d}) \models C(\bar{d}) \text{ as in Global Subsumption} \]

Non-Ground Global Subsumption
S

\[ \neg P(x) \lor Q(x) \]
\[ \neg Q(x) \lor S(x, y) \]
\[ P(x) \lor S(x, y) \]

C

\[ S(x, y) \lor Q(x) \]

Simplify first-order by purely ground reasoning!
Non-Ground Global Subsumption

\[
\begin{align*}
S & \quad C \\
\neg P(x) & \lor Q(x) \\
\neg Q(x) & \lor S(x, y) \\
P(x) & \lor S(x, y) \\
S & \quad \quad C \\
\neg P(a) & \lor Q(a) \\
\neg Q(a) & \lor S(a, b) \\
P(a) & \lor S(a, b) \\
\end{align*}
\]

Simplify first-order by purely ground reasoning!
Non-Ground Global Subsumption

\[
\begin{align*}
S & \\
\neg P(x) \lor Q(x) \\
\neg Q(x) \lor S(x, y) \\
P(x) \lor S(x, y) \\
\hline
S_{gr} & \\
\neg P(a) \lor Q(a) \\
\neg Q(a) \lor S(a, b) \\
P(a) \lor S(a, b)
\end{align*}
\]

\[
\begin{align*}
C & \\
S(x, y) \lor Q(x) \\
\hline
C_{gr} & \\
S(a, b) \lor Q(a)
\end{align*}
\]

Simplify first-order by purely ground reasoning!
Non-Ground Global Subsumption

\[
S \\
\neg P(x) \lor Q(x) \\
\neg Q(x) \lor S(x, y) \\
P(x) \lor S(x, y)
\]

\[
S_{gr} \\
\neg P(a) \lor Q(a) \\
\neg Q(a) \lor S(a, b) \\
P(a) \lor S(a, b)
\]

\[
C \\
S(x, y) \lor Q(x)
\]

\[
C_{gr} \\
S(a, b) \lor Q(a)
\]

Simplify first-order by purely ground reasoning!
Non-Ground Global Subsumption

\[ \neg P(x) \lor Q(x) \]

\[ \neg Q(x) \lor S(x, y) \]

\[ P(x) \lor S(x, y) \]

\[ S \]

\[ C \]

\[ S(x, y) \lor Q(x) \]

\[ \neg P(a) \lor Q(a) \]

\[ \neg Q(a) \lor S(a, b) \]

\[ P(a) \lor S(a, b) \]

\[ S_{gr} \]

\[ C_{gr} \]

\[ S(a, b) \lor Q(a) \]

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 $\sigma$ a grounding substitution

$$\left( A(a) \lor B(x) \right) \cdot [b/x]$$

Represents: ground clause $C\sigma$

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

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

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

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

Slogan: more specific representations take priority over less specific ones

Ex: $(p(a) \lor q(z)) \cdot [b/z] \prec (p(y) \lor q(z)) \cdot [a/y, b/z]$
Finer-grained control: replace ground clauses with ground closures.

Closure, a closure is a pair \( C \cdot \sigma \),
where \( C \) is a clause and \( \sigma \) a grounding substitution

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

Represents: ground clause \( C\sigma \)

\[
A(a) \lor B(b)
\]

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

\( C\theta \cdot \tau \prec C \cdot \sigma \) if

▶ \( C\sigma = C\theta\tau \), and

▶ \( \theta \) properly instantiates \( C \)

Slogan: more specific representations take priority over less specific ones

Ex: \( (p(a) \lor q(z)) \cdot [b/z] \prec (p(y) \lor q(z)) \cdot [a/y, b/z] \)
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 \( \sigma \) a grounding substitution

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

Represents: ground clause \( C\sigma \)

\[
A(a) \lor B(b)
\]

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

\( C\theta \cdot \tau \prec C \cdot \sigma \) if

\begin{itemize}
  \item \( C\sigma = C\theta\tau \), and
  \item \( \theta \) properly instantiates \( C \)
\end{itemize}

Slogan: more specific representations take priority over less specific ones

Ex: \( (p(a) \lor q(z)) \cdot [b/z] \prec (p(y) \lor q(z)) \cdot [a/y, b/z] \)
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
  - mismatching constraints
- redundancy for inferences
Dismatching Constraints [Korovin (IJCAR’08, vol. HG’13)]

Example:

\[ \begin{align*}
    p(x) \vee \neg q(f(x)) & \quad (1) \\
    p(f(x)) \vee \neg q(f(f(x))) & \quad (2) \\
    q(f(f(a))) & \quad (3)
\end{align*} \]

Then the inference between (1) and (2) 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.

\[ p(x) \vee \neg q(f(x)) \mid x \triangleleft_{ds} f(x) \]

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:

\[ \begin{align*}
  p(x) \lor \neg q(f(x)) & \quad (1) \\
  p(f(x)) \lor \neg q(f(f(x))) & \quad (2) \\
  q(f(f(a))) & \quad (3)
\end{align*} \]

Then the inference between (1) and (2) is redundant!

Why? the conclusion is represented twice

\[ p(f(a)) \lor \neg q(f(f(a))) \]

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

This can be represented as a dismatching constraint.

\[ p(x) \lor \neg q(f(x)) \vdash x \prec_{ds} f(x) \]

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:

\[ p(x) \lor \neg q(f(x)) \]  
\[ p(f(x)) \lor \neg q(f(f(x))) \] (2)  
\[ q(f(f(a))) \] (3)

Then the inference between (1) and (2) is **redundant**!

**Why?** the conclusion is represented twice \( p(f(a)) \lor \neg q(f(f(a))) \)

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

This can be represented as a **dismatching constraint**.

\[ p(x) \lor \neg q(f(x)) \mid x \prec_{ds} f(x) \]

**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:

\[
\begin{align*}
p(x) \lor \neg q(f(x)) & \quad (1) \\
p(f(x)) \lor \neg q(f(f(x))) & \quad (2) \\
q(f(f(a))) & \quad (3)
\end{align*}
\]

Then the inference between (1) and (2) is **redundant**!

**Why?** the conclusion is represented twice

\[
p(f(a)) \lor \neg q(f(f(a)))
\]

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

This can be represented as a **dismatching constraint**.

\[
p(x) \lor \neg q(f(x)) \mid x \prec_{ds} f(x)
\]

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:

\[ p(x) \lor \neg q(f(x)) \quad (1) \]
\[ p(f(x)) \lor \neg q(f(f(x))) \quad (2) \]
\[ q(f(f(a))) \quad (3) \]

Then the inference between (1) and (2) is redundant!

Why? the conclusion is represented twice \[ p(f(a)) \lor \neg q(f(f(a))) \]
\[ p(f(x)) \lor \neg q(f(f(x))) \cdot [a/x] \prec p(x) \lor \neg q(f(x)) \cdot [f(a)/x] \]

This can be represented as a dismatching constraint.

\[ p(x) \lor \neg q(f(x)) \mid x \prec_{ds} f(x) \]

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:

\[ p(x) \lor \neg q(f(x)) \] (1)
\[ p(f(x)) \lor \neg q(f(f(x))) \] (2)
\[ q(f(f(a))) \] (3)

Then the inference between (1) and (2) is redundant!

**Why?** the conclusion is represented twice \[ p(f(a)) \lor \neg q(f(f(a))) \]
\[ p(f(x)) \lor \neg q(f(f(x))) \cdot [a/x] \prec p(x) \lor \neg q(f(x)) \cdot [f(a)/x] \]

This can be represented as a **dismatching constraint**.

\[ p(x) \lor \neg q(f(x)) \mid x \prec_{ds} f(x) \]

**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:

\[ p(x) \lor \neg q(f(x)) \] (1)
\[ p(f(x)) \lor \neg q(f(f(x))) \] (2)
\[ q(f(f(a))) \] (3)

Then the inference between (1) and (2) is redundant!

Why? the conclusion is represented twice \( p(f(a)) \lor \neg q(f(f(a))) \)
\[ p(f(x)) \lor \neg q(f(f(x))) \cdot [a/x] \prec p(x) \lor \neg q(f(x)) \cdot [f(a)/x] \]

This can be represented as a dismatching constraint.

\[ p(x) \lor \neg q(f(x)) \mid x \triangleleft_{ds} f(x) \]

How to make closures redundant? Instantiate!

Every proper instantiation inference makes closures redundant in the premise.
Dismatching Constraints [Korovin IJCAR'08, HG’13]

Example

\[ A(f(y)) \lor D_1 \quad \neg A(x) \lor C \]
\[ A(f^3(y)) \lor D_2 \]
\[ A(f^5(y)) \lor D_3 \]
\[ \ldots \]
\[ A(f^{i_n}(y)) \lor D_n \]

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

Premises inherit the constraints during instantiation inferences.
Example

\[ A(f(y)) \lor D_1 \quad \neg A(x) \lor C \mid x <_{ds} f(y) \]
\[ A(f^3(y)) \lor D_2 \quad \neg A(f(y)) \lor C \]
\[ A(f^5(y)) \lor D_3 \]
\[ \ldots \]
\[ A(f^{i_n}(y)) \lor D_n \]

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

Premises inherit the constraints during instantiation inferences.
Dismatching Constraints [Korovin IJCAR'08, HG’13]

Example

\[ \begin{align*}
A(f(y)) & \lor D_1 & \neg A(x) & \lor C \mid x \prec_{ds} f(y) \\
A(f^3(y)) & \lor D_2 & \neg A(f(y)) & \lor C \\
A(f^5(y)) & \lor D_3 \\
\cdots \\
A(f^n(y)) & \lor D_n
\end{align*} \]

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:
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:
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
    - mismatching 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
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:

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

where (i) \( \theta = \text{mgu}(s, s') \), (ii) \( s' \) is not a variable, (iii) \( s\sigma \succ t\sigma \), (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**

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

\[
\theta = \text{mgu}(l, l')
\]

**Instantiation?**

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

\[
\theta = \text{mgu}(l, l')
\]
**Equality Superposition vs Inst-Gen**

**Superposition**

\[
\begin{align*}
C \lor l & \simeq r & L[l'][\lor D] \\
(C \lor D \lor L[r])\theta
\end{align*}
\]

\[\theta = \text{mgu}(l, l')\]

**Instantiation?**

\[
\begin{align*}
C \lor l & \simeq r & L[l'][\lor D] \\
(C \lor l \simeq r)\theta & & (L[l'][\lor D])\theta
\end{align*}
\]

\[\theta = \text{mgu}(l, l')\]

Incomplete!
Superposition + Instantiation

\[
\begin{align*}
  f(h(x)) & \simeq c \\
  h(x) & \simeq x \\
  f(a) & \not\simeq c
\end{align*}
\]

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

\[ f(h(x)) \simeq c \]
\[ h(x) \simeq x \]
\[ f(a) \not\simeq c \]

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:

\[ h(x) \simeq x \quad f(h(y)) \simeq c \]
\[ \frac{f(x) \simeq c}{f(a) \not\simeq c} \]
\[ \frac{c \not\simeq c}{\Box} \]
Superposition + Instantiation

\[ f(h(x)) \equiv c \]
\[ h(x) \equiv x \]
\[ f(a) \not\equiv c \]

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:

\[
\begin{align*}
 h(x) &\equiv x \\
 f(h(x)) &\equiv c \\
 f(x) &\equiv c \\
 f(a) &\not\equiv c \\
 c &\not\equiv c
\end{align*}
\]

This set is ground unsatisfiable. 

\[ \square \]
Superposition \textbf{+} Instantiation

\[
\begin{aligned}
 f(h(x)) & \simeq c \\
h(x) & \simeq x \\
f(a) & \not\simeq c
\end{aligned}
\]

This set is \textbf{inconsistent} but the \textbf{contradiction is not deducible} by the inference system above.

The idea is to consider \textbf{proofs generated by unit superposition}:

\[
\frac{
 h(x) \simeq x \quad f(h(y)) \simeq c
}{
 f(x) \simeq c \quad [x/y]
}\]

\[
\frac{
 f(a) \not\simeq c \\
 c \not\simeq c
}{
 \square
}\]

Propagating substitutions: \{ \textit{h(a) } \simeq a; f(h(a)) \simeq c; f(a) \not\simeq c \}

\textit{ground unsatisfiable.}
Superposition + Instantiation

\[ f(h(x)) \simeq c \lor C_1(x, y) \]
\[ h(x) \simeq x \lor C_2(x, y) \]
\[ f(a) \not\simeq c \lor C_3(x, y) \]

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:

\[
\begin{align*}
  h(x) &\simeq x \quad f(h(y)) \simeq c \\
  f(x) &\simeq c \quad [x/y] \\
  f(a) &\not\simeq c \quad [a/x] \\
  c &\not\simeq c \\
\end{align*}
\]

Propagating substitutions: \{h(a) \simeq a; f(h(a)) \simeq c; f(a) \not\simeq c\}

ground unsatisfiable.
Superposition + Instantiation

\[
\begin{align*}
  f(h(x)) & \simeq c \lor C_1(x, y) & f(h(a)) & \simeq c \lor C_1(a, y) \\
  h(x) & \simeq x \lor C_2(x, y) & h(a) & \simeq a \lor C_2(a, y) \\
  f(a) & \not\simeq c \lor C_3(x, y) & f(a) & \not\simeq c \lor C_3(a, y)
\end{align*}
\]

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:

\[
\begin{align*}
  h(x) \simeq x \quad & f(h(y)) \simeq c \\
  f(x) \simeq c \quad & [x/y] \\
  f(a) \not\simeq c \quad & [a/x] \\
  c \not\simeq c \\
\end{align*}
\]

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

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

\[ S \rightarrow \bot \]

\[ S \rightarrow \bot \text{ UnSAT} \]

\[ S \rightarrow \bot \text{ SAT} \]

\[ I \models S \]

Semantic selection of literals \[ I \models L \]

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.

The diagram shows the process of Inst-Gen-Eq:

1. **f.-o. clauses** $S$
2. **Semantic selection** of literals $I \models L$:
   - $S \models SAT$
   - $I \models S$
3. **Ground Clauses** $S_{\bot}$
4. **Inst. gen. from UP proofs**
5. **$\bot : \overline{x} \rightarrow \bot$**
6. **$\bot \models UnSAT$**
7. **theorem proved**
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:

\[
\begin{align*}
(l \simeq r) \mid [D_1] & \quad L[l'] \mid [D_2] \\
\frac{L[r] \theta \mid [(D_1 \land D_2) \theta]}{(\theta)} & \quad (s \not\simeq t) \mid [D] \\
\end{align*}
\]

where (i) \( \theta = \text{mgu}(l, l') \); (ii) \( l' \) is not a variable; (iii) for some grounding substitution \( \sigma \), satisfying \((D_1 \land D_2) \theta, l \sigma \gg r \sigma \); (iv) \( \mu = \text{mgu}(s, t) \); (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:

\[
\frac{(l \sim r) \mid [D_1] \quad L[l'] \mid [D_2]}{L[r] \theta \mid [(D_1 \land D_2) \theta]} (\theta)
\]

\[
s \not\sim t \mid [D]
\]

\[
\square (\mu)
\]

where (i) \( \theta = \text{mgu}(l, l') \); (ii) \( l' \) is not a variable; (iii) for some grounding substitution \( \sigma \), satisfying \( (D_1 \land D_2) \theta \), \( l\sigma \succ r\sigma \); (iv) \( \mu = \text{mgu}(s, t) \); (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, Stickel LPAR’10]

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

Unit superposition with dismatching constraints:

\[
\frac{(l \simeq r) \mid [D_1] L[l'] \mid [D_2]}{L[r] \theta \mid [(D_1 \land D_2) \theta]} (\theta)
\]

\[
\frac{s \not\simeq t \mid [D]}{\square} (\mu)
\]

where (i) \(\theta = \text{mgu}(l, l')\); (ii) \(l'\) is not a variable; (iii) for some grounding substitution \(\sigma\), satisfying \((D_1 \land D_2) \theta, l\sigma \succ r\sigma\); (iv) \(\mu = \text{mgu}(s, t)\); (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 $\land$ in superposition, disjunction $\lor$ in merging

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

### Solved equational problems

- **Sets**
  - 2006: 193
  - OBDD: 1512

- **Trees**
  - 1983: 216
  - OBDD: 1512

- **OBDDs**
  - 1983: 344
  - 2006: 30
  - OBDD: 13

### Features

<table>
<thead>
<tr>
<th></th>
<th>Normal form</th>
<th>Precise elim.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Trees</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>OBDDs</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

[Korovin, Sticksel LPAR’10]
Theory instantiation
Theory instantiation [Ganzinger, Korovin LPAR'06]

f.-o. clauses $S$
theory $T$
Theory instantiation [Ganzinger, Korovin LPAR’06]

\[ \text{f.-o. clauses } S \quad \perp : \overline{x} \rightarrow \perp \quad \text{Ground Clauses } S_{\perp} \]
Theory instantiation [Ganzinger, Korovin LPAR'06]

\[
\begin{align*}
\text{f.-o. clauses } S \\
\text{theory } T \\
\downarrow : \overline{x} \rightarrow \bot \\
\text{Ground Clauses } S_{\bot} \\
S_{\bot} \text{ UnSAT} \\
\text{theorem proved}
\end{align*}
\]
Theory instantiation [Ganzinger, Korovin LPAR’06]

f.-o. clauses $S$

theory $T$

$\bot : \overline{x} \rightarrow \bot$

Ground Clauses $S_\bot$

$S_\bot$ UnSAT

theorem proved

$S_\bot$ SAT

$I_\bot \models_T S_\bot$

Semantic selection of literals $I_\bot \models_T \mathcal{L}_\bot$
Theory instantiation [Ganzinger, Korovin LPAR’06]

- f.-o. clauses $S$
  - theory $T$
  - $\bot : \bar{x} \rightarrow \bot$
  - $\bot \vdash \bot$

  $L_1 \lor C_1, \ldots, L_n \lor C_n$
  $\frac{(L_1 \lor C_1)\theta, \ldots, (L_n \lor C_n)\theta}{L_1\theta \bot \land \ldots \land L_n\theta \bot \vdash_T 0}$

- Ground Clauses $S_\bot$
  - $S_\bot$ UnSAT
  - $S_\bot$ SAT
  - $I_\bot \models_T S_\bot$

- Semantic selection of literals $I_\bot \models_T L_\bot$

- $L \vdash_T \square$

- Theorem proved
**Theory instantiation** [Ganzinger, Korovin LPAR’06]

- f.-o. clauses $S$
  - theory $T$
  - $\perp : \overline{x} \rightarrow \perp$
  - $S \perp : \overline{x} \rightarrow \perp$
  - $\overline{x} \rightarrow \perp$

- Ground Clauses $S \perp$
  - $S \perp : \overline{x} \rightarrow \perp$
  - $S \perp : \overline{x} \rightarrow \perp$
  - $S \perp : \overline{x} \rightarrow \perp$
  - $S \perp : \overline{x} \rightarrow \perp$
  - $S \perp : \overline{x} \rightarrow \perp$

- $S \perp \ \text{UnSAT}$
  - $S \perp \ \text{UnSAT}$
  - $S \perp \ \text{UnSAT}$
  - $S \perp \ \text{UnSAT}$

- Semantic selection of literals $I \perp \vdash_T S \perp$
  - $I \perp \vdash_T S \perp$
  - $I \perp \vdash_T S \perp$
  - $I \perp \vdash_T S \perp$

- $S \perp \ \text{SAT}$
  - $S \perp \ \text{SAT}$
  - $S \perp \ \text{SAT}$
  - $S \perp \ \text{SAT}$

- $L_1 \lor C_1, \ldots, L_n \lor C_n$
  - $(L_1 \lor C_1) \theta, \ldots, (L_n \lor C_n) \theta$
  - $L_1 \theta \perp \land \ldots \land L_n \theta \perp \vdash_T 0$

- $\mathcal{L} \vdash_T \square$
  - $\mathcal{L} \vdash_T \square$
  - $\mathcal{L} \vdash_T \square$
  - $\mathcal{L} \vdash_T \square$

- $S$ satisfiable
  - $S$ satisfiable
  - $S$ satisfiable
  - $S$ satisfiable
Theory instantiation

Conditions on completeness:

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

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

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

such that $\theta$ is a generalization of $\tau$ and $L_1\theta \bot, \ldots, L_n\theta \bot \models_T \Box$

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 $L_1\tau \land \ldots \land L_n\tau \models_T \Box$ for ground $\tau$. Then

\[
\frac{L_1, \ldots, L_n}{L_1\theta, \ldots, L_n\theta} \quad UC
\]

such that $\theta$ is a generalization of $\tau$ and $L_1\theta \perp, \ldots, L_n\theta \perp \vdash_T \Box$

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 $L_1\tau \land \ldots \land L_n\tau \models_T \Box$ for ground $\tau$. Then

\[
\frac{L_1, \ldots, L_n}{L_1\theta, \ldots, L_n\theta} \quad UC
\]

such that $\theta$ is a generalization of $\tau$ and $L_1\theta \perp, \ldots, L_n\theta \perp \vdash_T \Box$

Theorem. Theory instantiation is sound and complete under these conditions.
Evaluation
## CASC 2013 results

### General first-order (FOF) 300 problems

<table>
<thead>
<tr>
<th></th>
<th>Vampire</th>
<th>E</th>
<th>iProver</th>
<th>E-KRHyper</th>
<th>Prover9</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>281</td>
<td>249</td>
<td>167</td>
<td>122</td>
<td>119</td>
</tr>
<tr>
<td>time</td>
<td>12</td>
<td>29</td>
<td>12</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

### Effectively propositional 100 problems

<table>
<thead>
<tr>
<th></th>
<th>iProver</th>
<th>Vampire</th>
<th>PEPR</th>
<th>E</th>
<th>EKRHyper</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>81</td>
<td>47</td>
<td>43</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>time</td>
<td>27</td>
<td>15</td>
<td>26</td>
<td>50</td>
<td>27</td>
</tr>
</tbody>
</table>

### First-order satisfiability (FNT) 150 problems

<table>
<thead>
<tr>
<th></th>
<th>iProver</th>
<th>Paradox</th>
<th>CVC4</th>
<th>E</th>
<th>Nitrox</th>
<th>Vampire</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>122</td>
<td>99</td>
<td>96</td>
<td>79</td>
<td>79</td>
<td>78</td>
</tr>
<tr>
<td>time</td>
<td>52</td>
<td>2</td>
<td>25</td>
<td>20</td>
<td>29</td>
<td>30</td>
</tr>
</tbody>
</table>

Non-cyclic sorts for first-order satisfiability [Korovin FroCoS’13]
## CASC 2013 results

### General first-order (FOF) 300 problems

<table>
<thead>
<tr>
<th></th>
<th>Vampire</th>
<th>E</th>
<th>iProver</th>
<th>E-KRHyper</th>
<th>Prover9</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>281</td>
<td>249</td>
<td>167</td>
<td>122</td>
<td>119</td>
</tr>
<tr>
<td>time</td>
<td>12</td>
<td>29</td>
<td>12</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

### Effectively propositional 100 problems

<table>
<thead>
<tr>
<th></th>
<th>iProver</th>
<th>Vampire</th>
<th>PEPR</th>
<th>E</th>
<th>EKRHyper</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>81</td>
<td>47</td>
<td>43</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>time</td>
<td>27</td>
<td>15</td>
<td>26</td>
<td>50</td>
<td>27</td>
</tr>
</tbody>
</table>

### First-order satisfiability (FNT) 150 problems

<table>
<thead>
<tr>
<th></th>
<th>iProver</th>
<th>Paradox</th>
<th>CVC4</th>
<th>E</th>
<th>NitroX</th>
<th>Vampire</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>122</td>
<td>99</td>
<td>96</td>
<td>79</td>
<td>79</td>
<td>78</td>
</tr>
<tr>
<td>time</td>
<td>52</td>
<td>2</td>
<td>25</td>
<td>20</td>
<td>29</td>
<td>30</td>
</tr>
</tbody>
</table>

Non-cyclic sorts for first-order satisfiability [Korovin FroCoS’13]
# CASC 2013 results

## General first-order (FOF) 300 problems

<table>
<thead>
<tr>
<th></th>
<th>Vampire</th>
<th>E</th>
<th>iProver</th>
<th>E-KRHyper</th>
<th>Prover9</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>281</td>
<td>249</td>
<td>167</td>
<td>122</td>
<td>119</td>
</tr>
<tr>
<td>time</td>
<td>12</td>
<td>29</td>
<td>12</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

## Effectively propositional 100 problems

<table>
<thead>
<tr>
<th></th>
<th>iProver</th>
<th>Vampire</th>
<th>PEPR</th>
<th>E</th>
<th>EKRHyper</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>81</td>
<td>47</td>
<td>43</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>time</td>
<td>27</td>
<td>15</td>
<td>26</td>
<td>50</td>
<td>27</td>
</tr>
</tbody>
</table>

## First-order satisfiability (FNT) 150 problems

<table>
<thead>
<tr>
<th></th>
<th>iProver</th>
<th>Paradox</th>
<th>CVC4</th>
<th>E</th>
<th>Nitrox</th>
<th>Vampire</th>
</tr>
</thead>
<tbody>
<tr>
<td>prob</td>
<td>122</td>
<td>99</td>
<td>96</td>
<td>79</td>
<td>79</td>
<td>78</td>
</tr>
<tr>
<td>time</td>
<td>52</td>
<td>2</td>
<td>25</td>
<td>20</td>
<td>29</td>
<td>30</td>
</tr>
</tbody>
</table>

Non-cyclic sorts for first-order satisfiability [Korovin FroCoS’13]
Effectively propositional logic (EPR)
Effectively Propositional Logic (EPR)

EPR: No functions except constants: $P(x, y) \lor \neg Q(c, y)$
**Effectively Propositional Logic (EPR)**

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

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

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

Verification:

$$\forall A(wren_{h1} \land A = wraddrFunc \rightarrow \forall B(range_{[35,0]}(B) \rightarrow (imem'(A, B) \leftrightarrow iwrite(B)))).$$  

**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:

\[ \neg \text{mem}(a_1, x_1) \lor \neg \text{mem}(a_2, x_2) \lor \ldots \neg \text{mem}(a_n, x_n) \]
\[ \text{mem}(b_1, x_1) \lor \text{mem}(b_2, x_2) \lor \ldots \lor \text{mem}(b_n, x_n) \]

General lemmas:

\[ \neg a(x) \lor b(x) \quad \neg b(x) \lor \text{mem}(x, y) \]
\[ a(x) \lor \text{mem}(x, y) \]
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:

\[-\text{mem}(a_1, x_1) \lor \neg \text{mem}(a_2, x_2) \lor \ldots \lor \neg \text{mem}(a_n, x_n)\]
\[\text{mem}(b_1, x_1) \lor \text{mem}(b_2, x_2) \lor \ldots \lor \text{mem}(b_n, x_n)\]

General lemmas:

\[-a(x) \lor b(x) \quad \neg b(x) \lor \text{mem}(x, y)\]
\[a(x) \lor \text{mem}(x, y) \quad \text{mem}(x, y)\]
Properties of EPR

Direct reduction to SAT — exponential blow-up.

Satisfiability for EPR is \text{NEXPTIME}-complete.

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

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

Restricting instances:

\[ \neg \text{mem}(a_1, x_1) \lor \neg \text{mem}(a_2, x_2) \lor \ldots \neg \text{mem}(a_n, x_n) \]
\[ \text{mem}(b_1, x_1) \lor \text{mem}(b_2, x_2) \lor \ldots \lor \text{mem}(b_n, x_n) \]

General lemmas:

\[ \neg a(x) \lor b(x) \]
\[ \neg b(x) \lor \text{mem}(x, y) \]
\[ a(x) \lor \text{mem}(x, y) \]
\[ \text{mem}(x, y) \]

More expressive logics can speed up calculations!
Hardware verification

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 \( \text{Next}(S, S') \)

BMC can be encoded

\[
\begin{align*}
I(s_0) &; \neg P(s_k); & \text{initial and final states} \\
\forall S, S' (\text{Next}(S, S') \rightarrow T(S, S')); & \text{transition relation} \\
\text{Next}(s_0, s_1); \text{Next}(s_1, s_2); \ldots \text{Next}(s_{k-1}, s_k); & \text{next state relation}
\end{align*}
\]

- 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 $\text{Next}(S, S')$

BMC can be encoded

\[
I(s_0); \neg P(s_k); \quad \text{initial and final states}
\]

\[
\forall S, S' (\text{Next}(S, S') \rightarrow T(S, S')); \quad \text{transition relation}
\]

\[
\text{Next}(s_0, s_1); \text{Next}(s_1, s_2); \ldots \text{Next}(s_{k-1}, s_k); \quad \text{next state relation}
\]

- 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

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

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
  - \(-\) \(\text{inst}\_\text{pass}\_\text{queue1}\) \([-\text{conj}\_\text{dist}; +\text{conj}\_\text{symb}; -\text{num}\_\text{var}]\)
  - \(-\) \(\text{inst}\_\text{pass}\_\text{queue2}\) \([+\text{age}; -\text{num}\_\text{symb}]\)

**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 \(\bot\)-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
  - \(\text{inst\_pass\_queue1} \quad \{ -\text{conj\_dist}; +\text{conj\_symb}; -\text{num\_var} \}\)
  - \(\text{inst\_pass\_queue2} \quad \{ +\text{age}; -\text{num\_symb} \}\)

**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 \(\bot\)-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
  - $\text{inst\_pass\_queue1} [\text{conj\_dist}; \text{+conj\_symb}; \text{−num\_var}]$
  - $\text{inst\_pass\_queue2} [\text{+age}; \text{−num\_symb}]$

**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 $\bot$-grounding of conclusions to the SAT solver
Inst-Gen Loop

[Corovin (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, mismatching 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 $C$
- ...
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”:

urance \( C \) 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 $C = p(f(f(x))) \lor \neg p(g(y))$ 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 $fv(C)$. 
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.

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.
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.
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