Automated Theorem Proving

and some Applications to Verification

Laura Kovács
TU Vienna

Homework Exercises – Automated Theorem Proving, L. Kovács

Problem 1. Establish the unsatisfiability of the following set of four formulas, using the superposition inference system $\mathbb{SRF}$:

$$
\begin{array}{ll}
(1) & c = d \\
(2) & f(d) \neq d \vee a = b \\
(3) & f(c) = d \\
(4) & g(a, b) \neq g(b, a)
\end{array}
$$

Problem 2. The limit of an $\mathbb{I}$-inference process $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ is the set of formulas $\bigcup_i S_i$. In other words, the limit is the set of all derived formulas.

Suppose that we have an infinite inference process such that S_0 is unsatisfiable and we use the ground superposition inference system $\mathbb{SRF}$.

Question: does completeness of $\mathbb{SRF}$ imply that the limit of the process contains the empty clause? Justify your answer!

Homework Exercises – Automated Theorem Proving, L. Kovács

Problem 1. Establish the unsatisfiability of the following set of four formulas, using the superposition inference system $\mathbb{SRF}$:

$$
\begin{array}{ll}
(1) & c = d \\
(2) & f(d) \neq d \vee a = b \\
(3) & f(c) = d \\
(4) & g(a, b) \neq g(b, a)
\end{array}
\qquad
\begin{array}{lll}
(5) & f(d) = d & (1, 3) \quad \text{(superposition)}
\end{array}
$$

Homework Exercises – Automated Theorem Proving, L. Kovács

Problem 1. Establish the unsatisfiability of the following set of four formulas, using the superposition inference system $\mathbb{SRF}$:

(1)	$c = d$	(5)	$f(d) = d$	$(1,3)$	(superposition)
(2)	$f(d) \neq d \vee a = b$	(6)	$d \neq d \vee a = b$	$(2,5)$	(superposition)
(3)	$f(c) = d$				
(4)	$g(a,b) \neq g(b,a)$				

Homework Exercises – Automated Theorem Proving, L. Kovács

Problem 1. Establish the unsatisfiability of the following set of four formulas, using the superposition inference system $\mathbb{SRF}$:

(1)	$c = d$	(5)	$f(d) = d$	$(1,3)$	(superposition)
(2)	$f(d) \neq d \vee a = b$	(6)	$d \neq d \vee a = b$	$(2,5)$	(superposition)
(3)	$f(c) = d$	(7)	$a = b$	(6)	(equality resolution)
(4)	$g(a,b) \neq g(b,a)$				

Homework Exercises – Automated Theorem Proving, L. Kovács

Problem 1. Establish the unsatisfiability of the following set of four formulas, using the superposition inference system $\mathbb{SRF}$:

(1)	$c = d$	(5)	$f(d) = d$	$(1, 3)$	(superposition)
(2)	$f(d) \neq d \vee a = b$	(6)	$d \neq d \vee a = b$	$(2, 5)$	(superposition)
(3)	$f(c) = d$	(7)	$a = b$	(6)	(equality resolution)
(4)	$g(a, b) \neq g(b, a)$	(8)	$g(b, b) \neq g(b, b)$	$(4, 7)$	(superposition)

Homework Exercises – Automated Theorem Proving, L. Kovács

Problem 1. Establish the unsatisfiability of the following set of four formulas, using the superposition inference system $\mathbb{SRF}$:

(1)	$c = d$	(5)	$f(d) = d$	$(1,3)$	(superposition)
(2)	$f(d) \neq d \vee a = b$	(6)	$d \neq d \vee a = b$	$(2,5)$	(superposition)
(3)	$f(c) = d$	(7)	$a = b$	(6)	(equality resolution)
(4)	$g(a,b) \neq g(b,a)$	(8)	$g(b,b) \neq g(b,b)$	$(4,7)$	(superposition)
		(9)	$\square$	(8)	(equality resolution)

Problem 1. Establish the unsatisfiability of the following set of four formulas, using the superposition inference system $\mathbb{SRF}$:

(1)	$c = d$	(5)	$f(d) = d$	(1, 3)	(superposition)
(2)	$f(d) \neq d \vee a = b$	(6)	$d \neq d \vee a = b$	(2, 5)	(superposition)
(3)	$f(c) = d$	(7)	$a = b$	(6)	(equality resolution)
(4)	$g(a, b) \neq g(b, a)$	(8)	$g(b, b) \neq g(b, b)$	(4, 7)	(superposition)
		(9)	$\square$	(8)	(equality resolution)

Problem 2. The limit of an $\mathbb{I}$-inference process $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ is the set of formulas $\bigcup_i S_i$. In other words, the limit is the set of all derived formulas.

Suppose that we have an infinite inference process such that S_0 is unsatisfiable and we use the ground superposition inference system $\mathbb{SRF}$.

Question: does completeness of $\mathbb{SRF}$ imply that the limit of the process contains the empty clause? Justify your answer!

Saturation Algorithm: Fairness

Let $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ be an inference process with the limit S_∞.
The process is called fair if for every $\mathbb{I}$-inference

$$\frac{F_1 \quad \ldots \quad F_n}{F} \ ,$$

if $\{F_1, \ldots, F_n\} \subseteq S_\infty$, then there exists i such that $F \in S_i$.

$\mathbb{SRF}$ completeness, reformulated

Theorem The following conditions are equivalent.

1. $\mathbb{SRF}$ is complete.
2. Let S_0 be an unsatisfiable set of formulas and we have a fair $\mathbb{SRF}$-inference process with the initial set S_0. Then the limit of this inference process contains $\square$.

Outline

The Ground Superposition System $\mathbb{SRF}$

Binary resolution inferences can be represented using derivations in the superposition system.

The Ground Superposition System $\mathbb{SRF}$

- $\mathbb{SRF}$ is sound and complete.

The Ground Superposition System $\mathbb{SRF}$

- $\mathbb{SRF}$ is sound and complete.
- We use a fair saturation algorithm.

The Ground Superposition System $\mathbb{SRF}$

- $\mathbb{SRF}$ is sound and complete.
- We use a fair saturation algorithm.
- Can this inference system be used for efficient theorem proving?

The Ground Superposition System $\mathbb{SRF}$

- $\mathbb{SRF}$ is sound and complete.
- We use a fair saturation algorithm.
- Can this inference system be used for efficient theorem proving?

  Not really. It has too many inferences. For example, from the clause $f(a) = a$ we can derive any clause of the form

$$f^m(a) = f^n(a)$$

  where $m, n \geq 0$.
  Worst of all, the derived clauses can be much larger than the original clause $f(a) = a$.

The Ground Superposition System $\mathbb{SRF}$

- $\mathbb{SRF}$ is sound and complete.
- We use a fair saturation algorithm.
- We need to formalize the search for derivations

The Ground Superposition System $\mathbb{SRF}$

- $\mathbb{SRF}$ is sound and complete.
- We use a fair saturation algorithm.
- We need to formalize the search for derivations:
  - Literal selections;
  - Orderings;
  - Redundancy elimination.

The Ground Superposition System $\mathbb{SRF}$

- $\mathbb{SRF}$ is sound and complete.
- We use a fair saturation algorithm.
- We need to formalize the search for derivations:
  - Literal selections;
  - Orderings;
  - Redundancy elimination.

Recall:

- Literal: either an atom A or its negation $\neg A$.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$ of literals, where $n \geq 0$.
- Empty clause, denoted by $\square$: clause with 0 literals, that is, when $n = 0$.
- A formula in Clausal Normal Form (CNF): a conjunction of clauses.

Literal Selection Functions

A literal selection function selects literals in a clause.

- If C is non-empty, then at least one literal is selected in C.

Literal Selection Functions

A literal selection function selects literals in a clause.

- If C is non-empty, then at least one literal is selected in C.

We denote selected literals by underlining them, e.g.,

$$\underline{f(a) = a} \vee b = c$$

Completeness?

Superposition with selection may be incomplete.

Completeness?

Superposition with selection may be incomplete.

Consider this set of clauses:

$$
\begin{array}{ll}
(1) & q \neq \top \ \vee \ \underline{r = \top} \\
(2) & p \neq \top \ \vee \ \underline{q = \top} \\
(3) & r \neq \top \ \vee \ \underline{q \neq \top} \\
(4) & q \neq \top \ \vee \ \underline{p \neq \top} \\
(5) & p \neq \top \ \vee \ \underline{r \neq \top} \\
(6) & r \neq \top \ \vee \ \underline{p = \top} \\
(7) & r = \top \ \vee \ q = \top \ \vee \ \underline{p = \top}
\end{array}
$$

Completeness?

Superposition with selection may be incomplete.

Consider this set of clauses:

(1)	$q \neq \top \ \vee \ \underline{r = \top}$
(2)	$p \neq \top \ \vee \ \underline{q = \top}$
(3)	$r \neq \top \ \vee \ \underline{q \neq \top}$
(4)	$q \neq \top \ \vee \ \underline{p \neq \top}$
(5)	$p \neq \top \ \vee \ \underline{r \neq \top}$
(6)	$r \neq \top \ \vee \ \underline{p = \top}$
(7)	$r = \top \ \vee \ q = \top \ \vee \ \underline{p = \top}$

It is unsatisfiable:

(8)	$q = \top \ \vee \ p = \top$	$(6, 7)$
(9)	$q = \top$	$(2, 8)$
(10)	$r = \top$	$(1, 9)$
(11)	$q \neq \top$	$(3, 10)$
(12)	$\square$	$(9, 11)$

Completeness?

Superposition with selection may be incomplete.

Consider this set of clauses:

$$
\begin{array}{ll}
(1) & q \neq \top \ \vee\ \underline{r = \top} \\
(2) & p \neq \top \ \vee\ \underline{q = \top} \\
(3) & r \neq \top \ \vee\ \underline{q \neq \top} \\
(4) & q \neq \top \ \vee\ \underline{p \neq \top} \\
(5) & p \neq \top \ \vee\ \underline{r \neq \top} \\
(6) & r \neq \top \ \vee\ \underline{p = \top} \\
(7) & r = \top \ \vee\ q = \top \ \vee\ \underline{p = \top}
\end{array}
$$

It is unsatisfiable:

$$
\begin{array}{lll}
(8) & q = \top \ \vee\ p = \top & (6,7) \\
(9) & q = \top & (2,8) \\
(10) & r = \top & (1,9) \\
(11) & q \neq \top & (3,10) \\
(12) & \square & (9,11)
\end{array}
$$

However, any inference with selection applied to this set of clauses gives either a clause in this set, or a clause containing a clause in this set.

Term and Literal Orderings

Term orderings

We take any ordering $\succ$ on terms such that:

1. $\succ$ is well-founded: no infinite decreasing chain $l_0 \succ l_1 \succ l_2 \succ \ldots$;
2. $\succ$ is monotonic: if $l \succ r$, then $s[l] \succ s[r]$;
3. $\succ$ is stable under substitutions: if $l \succ r$, then $l\theta \succ r\theta$.

Term and Literal Orderings

Term orderings: simplification ordering

We take any ordering $\succ$ on terms such that:

1. $\succ$ is well-founded: no infinite decreasing chain $l_0 \succ l_1 \succ l_2 \succ \ldots$;
2. $\succ$ is monotonic: if $l \succ r$, then $s[l] \succ s[r]$;
3. $\succ$ is stable under substitutions: if $l \succ r$, then $l\theta \succ r\theta$.

In the sequel $\succ$ will always denote a simplification ordering.

Term and Literal Orderings

Term orderings: simplification ordering

We take any ordering $\succ$ on terms such that:

1. $\succ$ is well-founded: no infinite decreasing chain $l_0 \succ l_1 \succ l_2 \succ \ldots$;
2. $\succ$ is monotonic: if $l \succ r$, then $s[l] \succ s[r]$;
3. $\succ$ is stable under substitutions: if $l \succ r$, then $l\theta \succ r\theta$.

In the sequel $\succ$ will always denote a simplification ordering.

Note: For every term $s[l]$ and its proper subterm l, we have $l \not\succ s[l]$.

Term and Literal Orderings

Term orderings: simplification ordering

We take any ordering $\succ$ on terms such that:

1. $\succ$ is well-founded: no infinite decreasing chain $l_0 \succ l_1 \succ l_2 \succ \ldots$;
2. $\succ$ is monotonic: if $l \succ r$, then $s[l] \succ s[r]$;
3. $\succ$ is stable under substitutions: if $l \succ r$, then $l\theta \succ r\theta$.

In the sequel $\succ$ will always denote a simplification ordering.

Note: For every term $s[l]$ and its proper subterm l, we have $l \not\succ s[l]$.

Literal orderings on equalities

Equality atom comparison treats $s = t$ as the multiset $\dot{\{}s, t\dot{\}}$.

- $(s' = t') \succ_{lit} (s = t)$ if $\dot{\{}s', t'\dot{\}} \succ \dot{\{}s, t\dot{\}}$.
- $(s' \neq t') \succ_{lit} (s \neq t)$ if $\dot{\{}s', t'\dot{\}} \succ \dot{\{}s, t\dot{\}}$.

All non-equality literals are greater than all equality literals.

Orderings and Well-Behaved Selection Functions

A literal selection function is well-behaved if

- If all selected literals are positive, then all maximal (w.r.t. $\succ$) literals in C are selected.

In other words, either a negative literal is selected, or all maximal literals must be selected.

Completeness of Superposition with Selection

Superposition with selection is complete for every well-behaved selection function.

Completeness of Superposition with Selection

Superposition with selection is complete for every well-behaved selection function.

Consider our previous example:

(1) $q \neq \top \vee \underline{r = \top}$

(2) $p \neq \top \vee \underline{q = \top}$

(3) $r \neq \top \vee \underline{q \neq \top}$

(4) $q \neq \top \vee \underline{p \neq \top}$

(5) $p \neq \top \vee \underline{r \neq \top}$

(6) $r \neq \top \vee \underline{p = \top}$

(7) $r = \top \vee q = \top \vee \underline{p = \top}$

A well-behave selection function must satisfy:

1. $r \succ q$, because of (1)
2. $q \succ p$, because of (2)
3. $p \succ r$, because of (6)

There is no ordering that satisfies these conditions.

Ground Superposition Inference System $\mathbb{S}\mathrm{up}_{\succ,\sigma}$

Let σ be a literal selection function.

Superposition: (right and left)

$$\frac{\underline{l = r} \lor C \quad \underline{s[l] = t} \lor D}{s[r] = t \lor C \lor D} \text{ (Sup)}, \quad \frac{\underline{l = r} \lor C \quad \underline{s[l] \neq t} \lor D}{s[r] \neq t \lor C \lor D} \text{ (Sup)},$$

where (i) $l \succ r$, (ii) $s[l] \succ t$, (iii) $l = r$ is strictly greater than any literal in C, (iv) $s[l] = t$ is greater than or equal to any literal in D.

Equality Resolution:

$$\frac{\underline{s \neq s} \lor C}{C} \text{ (ER)},$$

Equality Factoring:

$$\frac{\underline{s = t} \lor s = t' \lor C}{s = t \lor t \neq t' \lor C} \text{ (EF)},$$

where (i) $s \succ t \succeq t'$; (ii) $s = t$ is greater than or equal to any literal in C.

Non-Ground Superposition Rule

Superposition:

$$\frac{\underline{l = r} \lor C \quad \underline{s[l'] = t} \lor D}{(s[r] = t \lor C \lor D)\theta} \text{ (Sup)}, \qquad \frac{\underline{l = r} \lor C \quad \underline{s[l'] \neq t} \lor D}{(s[r] \neq t \lor C \lor D)\theta} \text{ (Sup)},$$

where

1. θ is an mgu of l and l';
2. l' is not a variable;
3. $r\theta \not\succeq l\theta$;
4. $t\theta \not\succeq s[l']\theta$.
5. ...

Non-Ground Superposition Rule

Superposition:

$$\frac{\underline{l = r} \vee C \quad \underline{s[l'] = t} \vee D}{(s[r] = t \vee C \vee D)\theta}\ \text{(Sup)}, \qquad \frac{\underline{l = r} \vee C \quad \underline{s[l'] \neq t} \vee D}{(s[r] \neq t \vee C \vee D)\theta}\ \text{(Sup)},$$

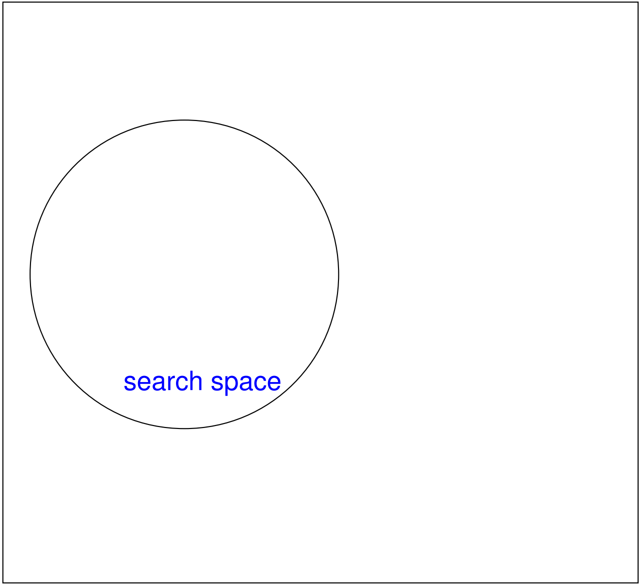
where

1. θ is an mgu of l and l';
2. l' is not a variable;
3. $r\theta \not\succeq l\theta$;
4. $t\theta \not\succeq s[l']\theta$.
5. ...

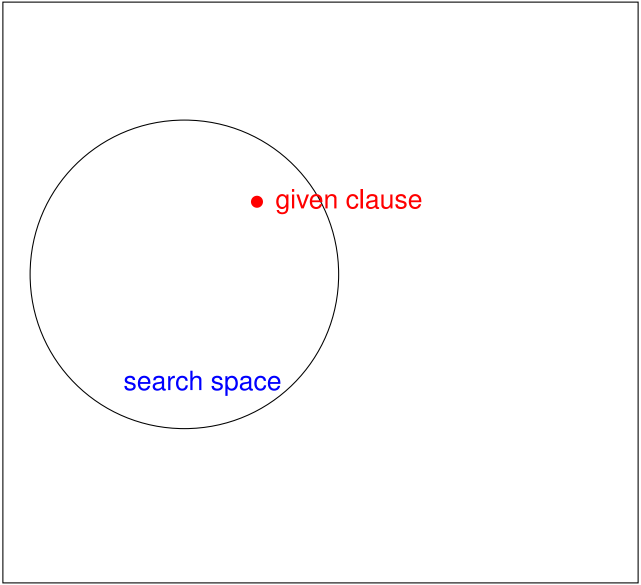
Observation:

- ordering is partial, hence conditions like $r\theta \not\succeq l\theta$;

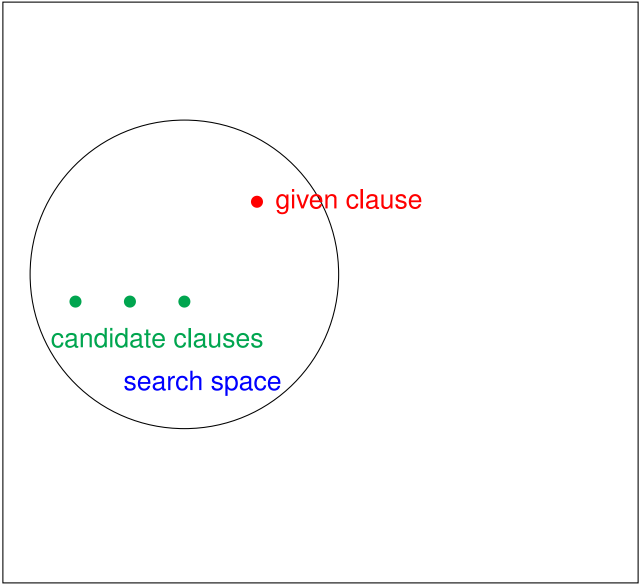
Fair Saturation Algorithms: Inference Selection by Clause Selection



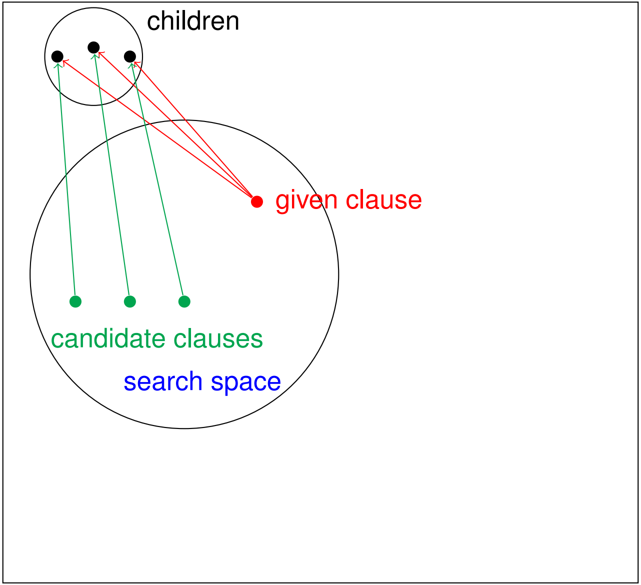
Fair Saturation Algorithms: Inference Selection by Clause Selection



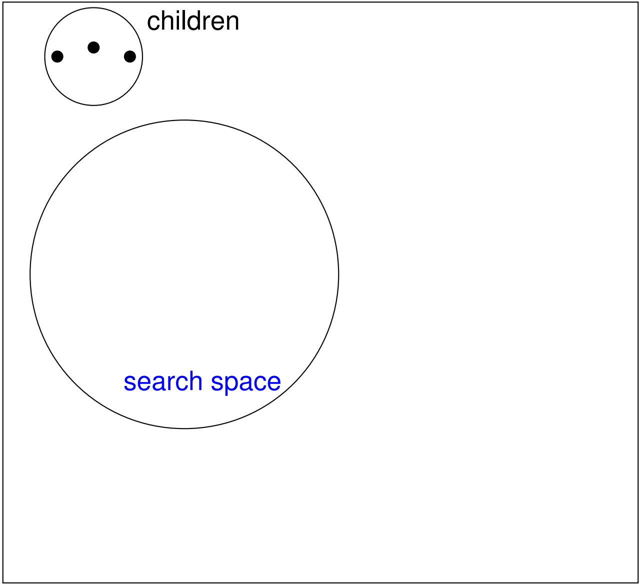
Fair Saturation Algorithms: Inference Selection by Clause Selection



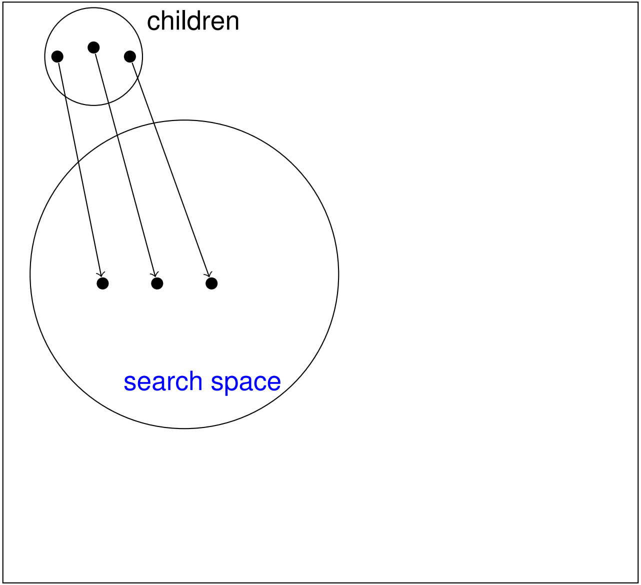
Fair Saturation Algorithms: Inference Selection by Clause Selection



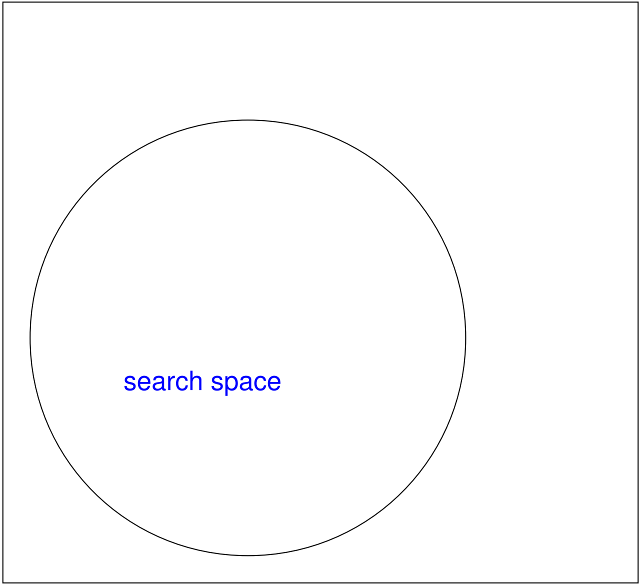
Fair Saturation Algorithms: Inference Selection by Clause Selection



Fair Saturation Algorithms: Inference Selection by Clause Selection



Fair Saturation Algorithms: Inference Selection by Clause Selection



Saturation Algorithm

Even when we implement inference selection by clause selection, there are too many inferences, especially when the search space grows.

Saturation Algorithm

Even when we implement inference selection by clause selection, there are too many inferences, especially when the search space grows.

Solution: only apply inferences to the selected clause and the previously selected clauses.

Saturation Algorithm

Even when we implement inference selection by clause selection, there are too many inferences, especially when the search space grows.

Solution: only apply inferences to the selected clause and the previously selected clauses.

Thus, the search space is divided in two parts:

- active clauses, that participate in inferences;
- passive clauses, that do not participate in inferences.

Redundancy

A clause $C \in S$ is called redundant in S if it is a logical consequence of clauses in S strictly smaller than C.

Examples

A tautology $p \vee \neg p \vee C$ is a logical consequence of the empty set of formulas:

$$\models p \vee \neg p \vee C,$$

therefore it is redundant.

Examples

A tautology $p \vee \neg p \vee C$ is a logical consequence of the empty set of formulas:

$$\models p \vee \neg p \vee C,$$

therefore it is redundant.

We know that C subsumes $C \vee D$. Note

$$C \vee D \succ C$$
$$C \models C \vee D$$

therefore subsumed clauses are redundant.

Redundant Clauses Can be Removed

In $\mathbb{SRF}$ redundant clauses can be removed from the search space.

Checking Redundancy

Suppose that the current search space S contains no redundant clauses. How can a redundant clause appear in the inference process?

Checking Redundancy

Suppose that the current search space S contains no redundant clauses. How can a redundant clause appear in the inference process?

Only when a new clause (a child of the selected clause and possibly other clauses) is added.

Classification of redundancy checks:

- The child is redundant;
- The child makes one of the clauses in the search space redundant.

Checking Redundancy

Suppose that the current search space S contains no redundant clauses. How can a redundant clause appear in the inference process?

Only when a new clause (a child of the selected clause and possibly other clauses) is added.

Classification of redundancy checks:

- The child is redundant; $\rightarrow$ forward simplification
- The child makes one of the clauses in the search space redundant. $\rightarrow$ backward simplification

Summary of a Proof by Vampire: Example from Algebra (recap)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

Summary of a Proof by Vampire: Example from Algebra (recap)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

- Proof by refutation;

Summary of a Proof by Vampire: Example from Algebra (recap)

```
Refutation found. Thanks to Tanya!
203. $false [subsumption resolution 202,14]
202. sP1(mult(sK,sK0)) [backward demodulation 188,15]
188. mult(X8,X9) = mult(X9,X8) [superposition 22,87]
87. mult(X2,mult(X1,X2)) = X1 [forward demodulation 71,27]
71. mult(inverse(X1),e) = mult(X2,mult(X1,X2)) [superposition 23,20]
27. mult(inverse(X2),e) = X2 [superposition 22,10]
23. mult(inverse(X4),mult(X4,X5)) = X5 [forward demodulation 18,9]
22. mult(X0,mult(X0,X1)) = X1 [forward demodulation 16,9]
20. e = mult(X0,mult(X1,mult(X0,X1))) [superposition 11,12]
18. mult(e,X5) = mult(inverse(X4),mult(X4,X5)) [superposition 11,10]
16. mult(e,X1) = mult(X0,mult(X0,X1)) [superposition 11,12]
15. sP1(mult(sK0,sK)) [inequality splitting 13,14]
14. ~sP1(mult(sK,sK0)) [inequality splitting name introduction]
13. mult(sK,sK0) != mult(sK0,sK) [cnf transformation 8]
12. e = mult(X0,X0) (0:5) [cnf transformation 4]
11. mult(mult(X0,X1),X2)=mult(X0,mult(X1,X2))[cnf transformation 3]
10. e = mult(inverse(X0),X0) [cnf transformation 2]
9. mult(e,X0) = X0 [cnf transformation 1]
8. mult(sK,sK0) != mult(sK0,sK) [skolemisation 7]
7. ? [X0,X1] : mult(X0,X1) != mult(X1,X0) [ennf transformation 6]
6. ~! [X0,X1] : mult(X0,X1) = mult(X1,X0) [negated conjecture 5]
5. ! [X0,X1] : mult(X0,X1) = mult(X1,X0) [input]
4. ! [X0] : e = mult(X0,X0)[input]
3. ! [X0,X1,X2] : mult(mult(X0,X1),X2) = mult(X0,mult(X1,X2))[input]
2. ! [X0] : e = mult(inverse(X0),X0) [input]
1. ! [X0] : mult(e,X0) = X0 [input]
```

- Proof by refutation;
- Each inference derives a new formula;
- Generating and simplifying inferences.

$\Big\}$ Fair saturation algorithm

Outline

The Superposition Inference System

Colored Proofs, Interpolation and Symbol Elimination

Sorts and Theories

Interpolation

Theorem

Let A, B be closed formulas and let $A \vdash B$.

Then there exists a formula I such that

1. $A \vdash I$ *and* $I \vdash B$*;*
2. *every symbol of* I *occurs both in* A *and* B*;*

Interpolation

Theorem

Let A, B be closed formulas and let $A \vdash B$.

Then there exists a formula I such that

1. $A \vdash I$ *and* $I \vdash B$*;*
2. *every symbol of I occurs both in A and B;*

Any formula I with this property is called an interpolant of A and B.

Interpolation has many uses in verification.

Interpolation

Theorem

Let A, B be closed formulas and let $A \vdash B$.

Then there exists a formula I such that

1. $A \vdash I$ *and* $I \vdash B$*;*
2. *every symbol of* I *occurs both in* A *and* B*;*

Any formula I with this property is called an interpolant of A and B.

Interpolation has many uses in verification.

When we deal with refutations rather than proofs and have an unsatisfiable set $\{A, B\}$, it is convenient to use reverse interpolants of A and B, that is, a formula I such that

1. $A \vdash I$ and $\{I, B\}$ is unsatisfiable;
2. every symbol of I occurs both in A and B.

Interpolation Through Colors

- There are three colors: blue, red and green.

Interpolation Through Colors

- There are three colors: blue, red and green.
- Each symbol (function or predicate) is colored in exactly one of these colors.

Interpolation Through Colors

- There are three colors: blue, red and green.
- Each symbol (function or predicate) is colored in exactly one of these colors.
- We have two formulas: A and B.
- Each symbol in A is either blue or green.
- Each symbol in B is either red or green.

Interpolation Through Colors

- There are three colors: blue, red and green.
- Each symbol (function or predicate) is colored in exactly one of these colors.
- We have two formulas: A and B.
- Each symbol in A is either blue or green.
- Each symbol in B is either red or green.
- We know that $\vdash A \rightarrow B$.
- Our goal is to find a green formula I such that
  1. $\vdash A \rightarrow I$;
  2. $\vdash I \rightarrow B$.

Local Derivations

A derivation is called local (well-colored) if each inference in it

$$\frac{C_1 \quad \cdots \quad C_n}{C}$$

either has no blue symbols or has no red symbols.
That is, one cannot mix blue and red in the same inference.

Local Derivations: Example

- A := $\forall x(x = a)$
- B := $c = b$
- Interpolant: $\forall x \forall y(x = y)$

Local Derivations: Example

- $A := \forall x(x = a)$
- $B := c = b$
- Interpolant: $\forall x \forall y(x = y)$

$$\dfrac{\dfrac{\dfrac{x = a}{c = a} \qquad \dfrac{x = a}{b = a}}{c = b} \qquad c \neq b}{\bot}$$

Local Derivations: Example

- $A := \forall x(x = a)$
- $B := c = b$
- Interpolant: $\forall x \forall y(x = y)$

Non-local proof

$$\dfrac{\dfrac{\dfrac{x = a}{c = a} \qquad \dfrac{x = a}{b = a}}{c = b} \qquad c \neq b}{\bot}$$

Local Derivations: Example

- $A := \forall x(x = a)$
- $B := c = b$
- Interpolant: $\forall x \forall y(x = y)$

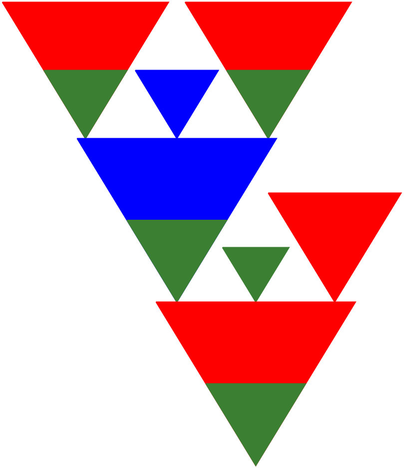
Non-local proof

$$\dfrac{\dfrac{\dfrac{x=a}{c=a} \qquad \dfrac{x=a}{b=a}}{c=b} \qquad c\neq b}{\bot}$$

Local Proof

$$\dfrac{\dfrac{\dfrac{x=a \qquad y=a}{x=y} \qquad c\neq b}{y\neq b}}{\bot}$$

Shape of a local derivation



Symbol Eliminating Inference

- At least one of the premises is not green.
- The conclusion is green.

$$\cfrac{\boxed{\cfrac{x = a \quad y = a}{x = y}} \qquad c \neq b}{\boxed{\cfrac{y \neq b}{\perp}}}$$

Extracting Interpolants from Local Proofs

Theorem (CADE'09)

Let Π be a local refutation. Then one can extract from Π in linear time a reverse interpolant I of A and B. This interpolant is ground if all formulas in Π are ground.

Extracting Interpolants from Local Proofs

Theorem (CADE'09)

Let Π be a local refutation. Then one can extract from Π in linear time a reverse interpolant I of A and B. This interpolant is ground if all formulas in Π are ground. This reverse interpolant is a boolean combination of conclusions of symbol-eliminating inferences of Π.

Extracting Interpolants from Local Proofs

Theorem (CADE'09)

Let Π be a local refutation. Then one can extract from Π in linear time a reverse interpolant I of A and B. This interpolant is ground if all formulas in Π are ground. This reverse interpolant is a boolean combination of conclusions of symbol-eliminating inferences of Π.

What is remarkable in this theorem:

- No restriction on the calculus (only soundness required) – can be used with theories.
- Can generate interpolants in theories where no good interpolation algorithms exist.

Extracting Interpolants from Local Proofs

Theorem (CADE'09)

Let Π be a local refutation. Then one can extract from Π in linear time a reverse interpolant I of A and B. This interpolant is ground if all formulas in Π are ground. This reverse interpolant is a boolean combination of conclusions of symbol-eliminating inferences of Π.

What is remarkable in this theorem:

- No restriction on the calculus (only soundness required) – can be used with theories.
- Can generate interpolants in theories where no good interpolation algorithms exist.

Further result: generate minimal interpolants wrt various measures. (POPL'12)

Symbol Elimination

Colored proofs can also be used for an interesting application. Suppose that we have a set of formulas in some language and want to derive consequences of these formulas in a subset of this language.

Symbol Elimination

Colored proofs can also be used for an interesting application. Suppose that we have a set of formulas in some language and want to derive consequences of these formulas in a subset of this language.

Then we declare the symbols to be eliminated colored and ask Vampire to output symbol-eliminating inferences.

Symbol Elimination

Colored proofs can also be used for an interesting application. Suppose that we have a set of formulas in some language and want to derive consequences of these formulas in a subset of this language.

Then we declare the symbols to be eliminated colored and ask Vampire to output symbol-eliminating inferences.

This technique was used in our experiments on automatic loop invariant generation (FASE'09).

Outline

How Vampire Proves Problems in Arithmetic

- adding theory axioms;
- evaluating expressions, when possible;
- (future) SMT solving.