Mechanizing first-order logic: Unification

Joris Roos

University of Wisconsin-Madison

Sommerakademie Leysin 2018

August 16, 2018

Overview

1. Introduction
2. Davis-Putnam
3. Unification

1 Introduction

2 Davis-Putnam

3 Unification

Review

Recall that we are studying the satisfiability of FOL formulas.

Review

Recall that we are studying the satisfiability of FOL formulas.
Last time we showed how to reduce a general FOL formula to an equisatisfiable formula without quantifiers (removing quantifiers by Skolemization).

Review

Recall that we are studying the satisfiability of FOL formulas.
Last time we showed how to reduce a general FOL formula to an equisatisfiable formula without quantifiers (removing quantifiers by Skolemization).

Theorem

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

Review

Recall that we are studying the satisfiability of FOL formulas.
Last time we showed how to reduce a general FOL formula to an equisatisfiable formula without quantifiers (removing quantifiers by Skolemization).

Theorem

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

Ground instances are propositional formulas obtained from substituting ground terms for free variables.

Review

Recall that we are studying the satisfiability of FOL formulas.
Last time we showed how to reduce a general FOL formula to an equisatisfiable formula without quantifiers (removing quantifiers by Skolemization).

Theorem

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

Ground instances are propositional formulas obtained from substituting ground terms for free variables.
Ground terms are terms made up only of function symbols and constant symbols of the language.

Review

Recall that we are studying the satisfiability of FOL formulas.
Last time we showed how to reduce a general FOL formula to an equisatisfiable formula without quantifiers (removing quantifiers by Skolemization).

Theorem

A quantifier-free formula F is satisfiable if and only if every finite set of ground instances is satisfiable.

Ground instances are propositional formulas obtained from substituting ground terms for free variables.
Ground terms are terms made up only of function symbols and constant symbols of the language.

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

1. Initialize $H = \top$.

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

1. Initialize $H = \top$.
2. Generate next ground instance G (a propositional formula).

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

1. Initialize $H = \top$.
2. Generate next ground instance G (a propositional formula).
3. Set $H := H \wedge G$.

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

1. Initialize $H = \top$.
2. Generate next ground instance G (a propositional formula).
3. Set $H := H \wedge G$.
4. Test if H is satisfiable (e.g. by converting to DNF).

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

1. Initialize $H = \top$.
2. Generate next ground instance G (a propositional formula).
3. Set $H := H \wedge G$.
4. Test if H is satisfiable (e.g. by converting to DNF).
    - If yes, go to (2).

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

1. Initialize $H = \top$.
2. Generate next ground instance G (a propositional formula).
3. Set $H := H \wedge G$.
4. Test if H is satisfiable (e.g. by converting to DNF).
   - If yes, go to (2).
   - If not, we are done.

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

1. Initialize $H = \top$.
2. Generate next ground instance G (a propositional formula).
3. Set $H := H \wedge G$.
4. Test if H is satisfiable (e.g. by converting to DNF).
   - If yes, go to (2).
   - If not, we are done.

If this terminates, then we proved that F is not satisfiable.

Naive theorem prover

Goal: show that quantifier-free formula F is not satisfiable.

1. Initialize $H = \top$.
2. Generate next ground instance G (a propositional formula).
3. Set $H := H \wedge G$.
4. Test if H is satisfiable (e.g. by converting to DNF).
    - If yes, go to (2).
    - If not, we are done.

If this terminates, then we proved that F is not satisfiable.

Naive theorem prover

How can we improve this?

Naive theorem prover

How can we improve this?

1. Use a better SAT procedure.

Naive theorem prover

How can we improve this?

1. Use a better SAT procedure.
   For example:
   - Use of DNF leads to combinatorial explosion because we keep joining formulas by $\wedge$.

Naive theorem prover

How can we improve this?

1. Use a better SAT procedure.
   For example:
   - Use of DNF leads to combinatorial explosion because we keep joining formulas by $\wedge$.
   - CNF/clause form is more natural (each step just adds a clause).

Naive theorem prover

How can we improve this?

1. Use a better SAT procedure.
   For example:
   - Use of DNF leads to combinatorial explosion because we keep joining formulas by $\wedge$.
   - CNF/clause form is more natural (each step just adds a clause).
   - Efficient algorithms to solve SAT for CNF formulas: Davis-Putnam, DPLL, ...

Naive theorem prover

How can we improve this?

1. Use a better SAT procedure.
   For example:
   - Use of DNF leads to combinatorial explosion because we keep joining formulas by $\wedge$.
   - CNF/clause form is more natural (each step just adds a clause).
   - Efficient algorithms to solve SAT for CNF formulas: Davis-Putnam, DPLL, ...
2. Substitute "clever" ground terms instead of a brute-force exhaustive search.

Naive theorem prover

How can we improve this?

1. Use a better SAT procedure.
   For example:
   - Use of DNF leads to combinatorial explosion because we keep joining formulas by $\wedge$.
   - CNF/clause form is more natural (each step just adds a clause).
   - Efficient algorithms to solve SAT for CNF formulas: Davis-Putnam, DPLL, ...
2. Substitute "clever" ground terms instead of a brute-force exhaustive search.
   One approach is *unification*.

1 Introduction

2 Davis-Putnam

3 Unification

We start with a formula in CNF represented as a list of clauses and want to decide if it is satisfiable.

We start with a formula in CNF represented as a list of clauses and want to decide if it is satisfiable.

This is done by iteratively applying three rules that do not change satisfiability.

We start with a formula in CNF represented as a list of clauses and want to decide if it is satisfiable.

This is done by iteratively applying three rules that do not change satisfiability.

We always assume that no clause contains both a literal and its negation, since $P \vee \neg P$ is a tautology.

Rule 1

If there is a clause that contains only a single literal (possibly negated) P, then

Rule 1

If there is a clause that contains only a single literal (possibly negated) P, then

- remove every clause containing P

Rule 1

If there is a clause that contains only a single literal (possibly negated) P, then

- remove every clause containing P
- remove every occurrence of $\neg P$ in other clauses

Rule 1

If there is a clause that contains only a single literal (possibly negated) P, then

- remove every clause containing P
- remove every occurrence of $\neg P$ in other clauses

Rule 2

If some literal P occurs either only unnegated or only negated, then remove every clause containing P.

Propositional resolution

Rule 3 is based on the following deduction rule: suppose we have two clauses of the form

$$P \vee A, \neg P \vee B$$

Propositional resolution

Rule 3 is based on the following deduction rule: suppose we have two clauses of the form

$$P \vee A, \neg P \vee B$$

where A, B are clauses and P a literal.

Propositional resolution

Rule 3 is based on the following deduction rule: suppose we have two clauses of the form

$$P \vee A, \neg P \vee B$$

where A, B are clauses and P a literal.
Then we can deduce the *resolvent clause*

$$A \vee B$$

Rule 3

Let P be a literal and suppose we have clauses

$$P \vee A_1, \ldots, P \vee A_n$$

for A_i clauses (not containing P, $\neg P$)

Rule 3

Let P be a literal and suppose we have clauses

$$P \vee A_1, \ldots, P \vee A_n$$

for A_i clauses (not containing P, $\neg P$)
and clauses

$$\neg P \vee B_1, \ldots, \neg P \vee B_m$$

for B_i clauses (not containing P, $\neg P$),

Rule 3

Let P be a literal and suppose we have clauses

$$P \vee A_1, \dots, P \vee A_n$$

for A_i clauses (not containing P, $\neg P$)
and clauses

$$\neg P \vee B_1, \dots, \neg P \vee B_m$$

for B_i clauses (not containing P, $\neg P$),
then we replace these by the clauses

$$A_i \vee B_j$$

for $i = 1, \dots, n, j = 1, \dots, m$ (and remove tautologies). This does not change satisfiability.

- Each rule reduces the number of literals.

- Each rule reduces the number of literals.
- If there is a nonempty clause, then one of the rules applies.

- Each rule reduces the number of literals.
- If there is a nonempty clause, then one of the rules applies.
- Consequently, we can keep applying the rules (we prefer Rules 1 and 2 when possible) and the procedure will terminate.

- Each rule reduces the number of literals.
- If there is a nonempty clause, then one of the rules applies.
- Consequently, we can keep applying the rules (we prefer Rules 1 and 2 when possible) and the procedure will terminate.
- Much faster than truth tables.

- Each rule reduces the number of literals.
- If there is a nonempty clause, then one of the rules applies.
- Consequently, we can keep applying the rules (we prefer Rules 1 and 2 when possible) and the procedure will terminate.
- Much faster than truth tables.
- This drastically improves our naive FOL theorem prover by avoiding combinatorial explosion owing to DNF.

- Each rule reduces the number of literals.
- If there is a nonempty clause, then one of the rules applies.
- Consequently, we can keep applying the rules (we prefer Rules 1 and 2 when possible) and the procedure will terminate.
- Much faster than truth tables.
- This drastically improves our naive FOL theorem prover by avoiding combinatorial explosion owing to DNF.
- *Catch:* Rule 3 may drastically increase the number of clauses.

- Each rule reduces the number of literals.
- If there is a nonempty clause, then one of the rules applies.
- Consequently, we can keep applying the rules (we prefer Rules 1 and 2 when possible) and the procedure will terminate.
- Much faster than truth tables.
- This drastically improves our naive FOL theorem prover by avoiding combinatorial explosion owing to DNF.
- *Catch:* Rule 3 may drastically increase the number of clauses.
- Improvement: Davis-Putnam-Logeman-Loveland (DPLL)

Example

$$[[P, Q, \neg R, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T], [R]]$$

Example

$$[[P, Q, \neg R, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T], [R]]$$

Apply Rule 1:

$$[[P, Q, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T]]$$

Example

$$[[P, Q, \neg R, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T], [R]]$$

Apply Rule 1:

$$[[P, Q, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T]]$$

Apply Rule 2:

$$[[P, Q, \neg S], [\neg P, \neg Q, S]]$$

Example

$$[[P, Q, \neg R, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T], [R]]$$

Apply Rule 1:

$$[[P, Q, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T]]$$

Apply Rule 2:

$$[[P, Q, \neg S], [\neg P, \neg Q, S]]$$

Apply Rule 3:

$$[[Q, \neg S, \neg Q, S]]$$

Example

$$[[P, Q, \neg R, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T], [R]]$$

Apply Rule 1:

$$[[P, Q, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T]]$$

Apply Rule 2:

$$[[P, Q, \neg S], [\neg P, \neg Q, S]]$$

Apply Rule 3:

$$[[Q, \neg S, \neg Q, S]]$$

Remove tautology:

$$[]$$

Example

$$[[P, Q, \neg R, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T], [R]]$$

Apply Rule 1:

$$[[P, Q, \neg S], [\neg P, \neg Q, S], [P, \neg Q, T]]$$

Apply Rule 2:

$$[[P, Q, \neg S], [\neg P, \neg Q, S]]$$

Apply Rule 3:

$$[[Q, \neg S, \neg Q, S]]$$

Remove tautology:

$$[]$$

No clauses left, so formula is satisfiable.

1 Introduction

2 Davis-Putnam

3 Unification

Idea

Consider the following quantifier-free FOL formula in clause form:

$$[[P(x, f(y))], [Q(x, y), \neg P(g(z), w)]]$$

Idea

Consider the following quantifier-free FOL formula in clause form:

$$[[P(x, f(y))], [Q(x, y), \neg P(g(z), w)]]$$

Substitute $x \mapsto g(z)$ and $w \mapsto f(y)$.

Idea

Consider the following quantifier-free FOL formula in clause form:

$$[[P(x, f(y))], [Q(x, y), \neg P(g(z), w)]]$$

Substitute $x \mapsto g(z)$ and $w \mapsto f(y)$.

$$[[P(g(z), f(y))], [Q(g(z), y), \neg P(g(z), f(y)]]$$

Idea

Consider the following quantifier-free FOL formula in clause form:

$$[[P(x, f(y))], [Q(x, y), \neg P(g(z), w)]]$$

Substitute $x \mapsto g(z)$ and $w \mapsto f(y)$.

$$[[P(g(z), f(y))], [Q(g(z), y), \neg P(g(z), f(y)]]$$

Then by resolution we may add the clause

$$Q(g(z), y)$$

Idea

Consider the following quantifier-free FOL formula in clause form:

$$[[P(x, f(y))], [Q(x, y), \neg P(g(z), w)]]$$

Substitute $x \mapsto g(z)$ and $w \mapsto f(y)$.

$$[[P(g(z), f(y))], [Q(g(z), y), \neg P(g(z), f(y)]]$$

Then by resolution we may add the clause

$$Q(g(z), y)$$

(This is still a FOL formula with free variables!)

Unifiers

Definition

An *instantiation* σ is a map assigning a term to each variable symbol.

Unifiers

Definition

An *instantiation* σ is a map assigning a term to each variable symbol.

(By structural induction we can uniquely extend σ to a map on the set of terms which we also denote by σ.)

Unifiers

Definition

An *instantiation* σ is a map assigning a term to each variable symbol.

(By structural induction we can uniquely extend σ to a map on the set of terms which we also denote by σ.)

Definition

Let S be a set of pairs of terms. An instantiation σ is a *unifier* of S if

$$\sigma(s) = \sigma(t)$$

for all $(s, t) \in S$.

MGU

Definition

If σ, σ' are instantiations, we say that σ is *more general* than σ' if there exists an instantiation δ such that $\sigma' = \delta \circ \sigma$.

MGU

Definition

If σ, σ' are instantiations, we say that σ is *more general* than σ' if there exists an instantiation δ such that $\sigma' = \delta \circ \sigma$.

Definition

A unifier is called a *most general unifier (MGU)* if it is more general than every other unifier.

MGU

Definition

If σ, σ' are instantiations, we say that σ is *more general* than σ' if there exists an instantiation δ such that $\sigma' = \delta \circ \sigma$.

Definition

A unifier is called a *most general unifier (MGU)* if it is more general than every other unifier.

- A MGU is a unifier that is "as simple as possible".

MGU

Definition

If σ, σ' are instantiations, we say that σ is *more general* than σ' if there exists an instantiation δ such that $\sigma' = \delta \circ \sigma$.

Definition

A unifier is called a *most general unifier (MGU)* if it is more general than every other unifier.

- A MGU is a unifier that is "as simple as possible".
- If a unifier exists, then a MGU exists and there is an algorithm to compute it.

MGU

Definition

If σ, σ' are instantiations, we say that σ is *more general* than σ' if there exists an instantiation δ such that $\sigma' = \delta \circ \sigma$.

Definition

A unifier is called a *most general unifier (MGU)* if it is more general than every other unifier.

- A MGU is a unifier that is "as simple as possible".
- If a unifier exists, then a MGU exists and there is an algorithm to compute it.
- MGUs are not necessarily unique.

MGU

Definition

If σ, σ' are instantiations, we say that σ is *more general* than σ' if there exists an instantiation δ such that $\sigma' = \delta \circ \sigma$.

Definition

A unifier is called a *most general unifier (MGU)* if it is more general than every other unifier.

- A MGU is a unifier that is "as simple as possible".
- If a unifier exists, then a MGU exists and there is an algorithm to compute it.
- MGUs are not necessarily unique.

Examples

Example 1. Let $S = \{(x+1, y)\}$

Examples

Example 1. Let $S = \{(x+1, y)\}$
Then $\sigma : y \mapsto x + 1$ is a MGU.

Examples

Example 1. Let $S = \{(x + 1, y)\}$
Then $\sigma : y \mapsto x + 1$ is a MGU.
$\sigma' : x \mapsto 1, y \mapsto 1 + 1$ is a unifier, but not a MGU.

Examples

Example 1. Let $S = \{(x+1, y)\}$
Then $\sigma : y \mapsto x + 1$ is a MGU.
$\sigma' : x \mapsto 1, y \mapsto 1 + 1$ is a unifier, but not a MGU.

Example 2. Let $S = \{(x, f(x))\}$.

Examples

Example 1. Let $S = \{(x+1, y)\}$
Then $\sigma : y \mapsto x+1$ is a MGU.
$\sigma' : x \mapsto 1, y \mapsto 1+1$ is a unifier, but not a MGU.

Example 2. Let $S = \{(x, f(x))\}$.
Then S has no unifiers.

FOL resolution

We can use this to build an improved FOL theorem prover by combining unification with resolution.

FOL resolution

We can use this to build an improved FOL theorem prover by combining unification with resolution.

We keep forming (unified) resolvents of clauses until we derive the empty clause.

FOL resolution

We can use this to build an improved FOL theorem prover by combining unification with resolution.

We keep forming (unified) resolvents of clauses until we derive the empty clause.
One can show that this always terminates if the original formula was not satisfiable.

FOL resolution

We can use this to build an improved FOL theorem prover by combining unification with resolution.

We keep forming (unified) resolvents of clauses until we derive the empty clause.
One can show that this always terminates if the original formula was not satisfiable.
(Example on the board)

FOL resolution

We can use this to build an improved FOL theorem prover by combining unification with resolution.

We keep forming (unified) resolvents of clauses until we derive the empty clause.
One can show that this always terminates if the original formula was not satisfiable.
(Example on the board)

Further directions

- Tableaux
- Subsumption and replacement
- Linear resolution
- Model elimination
- $\ldots$

References

John Harrison. *Handbook of Practical Logic and Automated Reasoning.* (Cambridge, 2009)