IN3070/4070 – Logic – Autumn 2020
Lecture 8: First-order Resolution

Martin Giese

8th October 2020

Department of Informatics
University of Oslo

Today's Plan

- Reminder: Clausal Form Translations
- Reminder: Propositional Resolution
- Reminder: Unification
- First-Order Resolution
- Soundness and Completeness
- Compactness
- Summary

Outline

Translation into Clausal Form – Example

Example: $\forall x\, \exists y\, p(x,y) \rightarrow \exists y\, \forall x\, p(x,y)$

Try to prove this formula based on refutation in CNF

- negate the formula: $\neg(\forall x\, \exists y\, p(x,y) \rightarrow \exists y\, \forall x\, p(x,y))$
- Rename bound variables: $\neg(\forall x\, \exists y\, p(x,y) \rightarrow \exists w\, \forall z\, p(z,w))$
- Eliminate implication $\rightarrow$: $\neg(\neg \forall x\, \exists y\, p(x,y) \vee \exists w\, \forall z\, p(z,w))$
- Push negation inwards: $\forall x\, \exists y\, p(x,y) \wedge \forall w\, \exists z\, \neg p(z,w)$
- Skolemize, i.e., replace $\exists$: $\forall x\, p(x,f(x)) \wedge \forall w\, \neg p(g(w),w)$
- Write in clausal form : $\{\{p(x,f(x))\},\{\neg p(g(w),w)\}\}$

Outline

- Reminder: Clausal Form Translations
- Reminder: Propositional Resolution
- Reminder: Unification
- First-Order Resolution
- Soundness and Completeness
- Compactness
- Summary

Reminder: The Resolution Rule

The resolution calculus is a refutation procedure.

- in order to determine whether a formula F (in clausal form) is valid, we check whether $\neg F$ is unsatisfiable

Definition 2.1 (Complementary Literal).

The complementary literal $\overline{L}$ of a literal L is A if L is of the form $\neg A$, otherwise it is $\neg L$.

Definition 2.2 (Resolution Rule).

Let C_1, C_2 be clauses with $L \in C_1$ and $\overline{L} \in C_2$. The resolvent C' of C_1 and C_2 is $(C_1 \backslash \{L\}) \cup (C_2 \backslash \{\overline{L}\})$. C_1 and C_2 are the parents of C'.

- the resolution rule maintains satisfiability: If $\mathcal{I} \models C_1$ and $\mathcal{I} \models C_2$ then $\mathcal{I} \models C'$
- if a set of clauses S is satisfiable and $C_1, C_2 \in S$, then $S \cup \{C'\}$ is satisfiable.

The Resolution Rule – Example

Example: Let $C_1 = \{a, b, \neg c\}$ and $C_2 = \{b, c, \neg e\}$.

$$\{a, b, \neg c\} \quad \{b, c, \neg e\}$$
$$\searrow \quad \swarrow$$
$$\{a, b, \neg e\}$$

The resolvent of C_1 and C_2 is $\{a, b, \neg e\}$.

Observations:

- if $\{a, b, \neg c\}$ and $\{b, c, \neg e\} \equiv (a \vee b \vee \neg c) \wedge (b \vee c \vee \neg e)$ are satisfiable, then $(a \vee b)$ is satisfiable (if c is true) or $(b \vee \neg e)$ is satisfiable (if c is false); hence $(a \vee b \vee \neg e)$ is satisfiable
- if resolvent is unsatisfiable, then parents are unsatisfiable
- the empty clauses $\{\,\}$ is unsatisfiable
- goal: derive empty clause $\{\,\}$

The Resolution Calculus

- a set of clauses is unsatisfiable iff the empty clause can be derived
- a clause C is true iff at least one of its literals is true; if there is no literal in C, then C is false and every set of clauses (in CNF) that contains C is false, i.e.unsatisfiable

Definition 2.3 (Resolution Procedure).

Given a set of clauses S.

1. *apply the resolution rule to a pair of clauses $\{C_1, C_2\} \subseteq S$ that has not been chosen before; let C' be the resolvent*
2. $S' := S \cup \{C'\}$, $S := S'$
3. *if $C' = \{\}$, then output "unsatisfiable";*
   if all possible resolvents have been considered, then output "satisfiable"; otherwise continue with 1.

Outline

- Reminder: Clausal Form Translations
- Reminder: Propositional Resolution
- Reminder: Unification
- First-Order Resolution
- Soundness and Completeness
- Compactness
- Summary

Unification

- Motivation: try refuting the following

$$\{\ \{p(x,b)\},\ \{\neg p(a,y)\}\ \}$$

- Remember: these mean

$$\forall x\, p(x,b) \quad \text{and} \quad \forall y\, \neg p(a,y)$$

- Should be OK to instantiate x with a and y with b
- Giving

$$\{\ \{p(a,b)\},\ \{\neg p(a,b)\}\ \}$$

- Which can be resolved to $\Box$

Unification problem

Let s and t be terms. Find *all* substitutions that make s and t syntactically equal, i.e. all σ with $\sigma(s) = \sigma(t)$.

- A substitution that makes s and t syntactically equal is called a unifier for s and t.
- To terms are unifiable if they have a unifier.

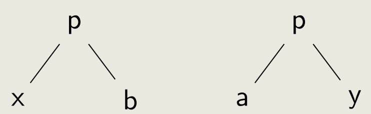
Examples

Are $f(x)$ and $f(a)$ unifiable?

Yes. We see that $\sigma = \{x \backslash a\}$ is a *unifier*: $\sigma(f(x)) = f(a)$

Are $p(x,b)$ and $p(a,y)$ unifiable?

Easier to see if we write terms as *trees*:

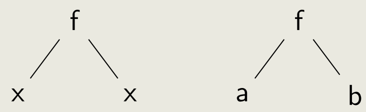


- The root symbols are the same.
- The left children are different, but can be unified with $\{x \backslash a\}$.
- The right children are different, but can be unified with $\{y \backslash b\}$.

Are $f(a,b)$ and $g(a,b)$ unifiable?

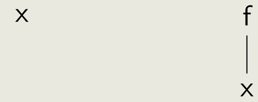
- The root symbols are different, and can *not* be unified!

Are $f(x, x)$ and $f(a, b)$ unifiable?



- The root symbols are equal.
- The left children are different, but can be unified with $\{x \backslash a\}$.
- We must apply $\{x \backslash a\}$ to x in both branches.
- The right children are now different, and can *not* be unified!

Are x and $f(x)$ unifiable?



- The root symbols are different, but can be unified by $\{x \backslash f(x)\}$.
- We also have to apply $\{x \backslash f(x)\}$ on x in the right tree.
- The symbols x and f are different.
- If we unify with $\{f(x)/x\}$, we have to replace x in the right tree again.
- This continues indefinitely

Unification

Generally:

- Two *distinct* constant or function symbols are not unifiable.
- A variable x is not unifiable with a term that *contains* x.

- We will define a unification algorithm, that finds all unifiers for two terms.
- Problem: Two terms can potentially have infinitely many unifiers. We can't compute all of them!
- Solution: Find a represetative σ for the set of unifiers, such that all other unifiers can be constructed from σ.
- Such a unifier is known as a most general unifier.

More General Substitution

Definition 3.1 (More General Substitution).

Let σ_1 and σ_2 be substitutions. We say that σ_2 is more general than σ_1 if there exists a subsitution τ such that $\sigma_1 = \tau\sigma_2$.

Is $\{x \backslash f(y)\}$ more general than $\{x \backslash f(a), y \backslash a\}$?

Yes, since $\{x \backslash f(a), y \backslash a\} = \{y \backslash a\}\{x \backslash f(y)\}$.

Is $\{x \backslash f(a)\}$ more general than $\{x \backslash f(y)\}$?

No, because there is no substitution τ such that $\{x \backslash f(y)\} = \tau\{x \backslash f(a)\}$.

Is $\{x \backslash f(y)\}$ more general than $\{x \backslash f(y)\}$

Yes, since $\{x \backslash f(y)\} = \{\}\{x \backslash f(y)\}$, where $\{\}$ is the identity substitution.

Most General Unifiers

Definition 3.2 (Unifier, Most General Unifier).

Let s and t be terms. A substitution σ is

- *a unifier for s and t if $\sigma(s) = \sigma(t)$.*
- *a most general unifier (mgu) for s and t if*
  - *it is a unifier for s and t, and*
  - *it is more general than any other unifiers for s and t.*

We say that s and t are unifiable if they have a unifier.

Let $s = f(x)$ and $t = f(y)$.

- $\sigma_1 = \{x\backslash a, y\backslash a\}$ is a unifier for s and t
- $\sigma_2 = \{x\backslash y\}$ and $\sigma_3 = \{y\backslash x\}$ are also unifiers for s and t
- σ_2 and σ_3 are the most general unifiers for s and t

Uniqueness "up to variable renaming"

Proposition 3.1.

If σ_1 and σ_2 are most general unifiers for two terms s and t, then there is a variable renaming η such that $\eta\sigma_1 = \sigma_2$.

- We leave out the proof.

Unification Algorithm

Algoritm: unify(t_1, t_2)

$\sigma := \epsilon$;
while ($\sigma(t_1) \neq \sigma(t_2)$) **do**
  choose a critical pair $\langle k_1, k_2\rangle$ for $\sigma(t_1), \sigma(t_2)$;
  if (neither k_1 nor k_2 are variables) **then**
    return *"not unifiable"*;
  end if
  x := the one of k_1, k_2 that is a variable (if both are, choose one)
  t := the one of k_1, k_2 that is not x;
  if (x occurs in t) **then**
    return *"not unifiable"*;
  end if
  $\sigma := \{x\backslash t\}\sigma$;
end while
return σ;

Properties of the Unification Algorithm

- If the terms t_1 and t_2 are unifiable, the algorithm returns a most general unifier for t_1 and t_2.
- The mgu is representative for all other unifiers of t_1 and t_2.
- If t_1 and t_2 are not unifiable, the algorithm returns *"not unifiable"*.

Outline

The First-Order Resolution Calculus

The resolution rule is generalized by performing unification as part of the rule and an additional factorization rule is added.

Definition 4.1 (First-Order Resolution Calculus).

$$\frac{}{C_1, ..., \{\}, ..., C_n}\ \textit{axiom}$$

$$\frac{C_1, ..., C_i \cup \{L_1\}, ..., C_j \cup \{L_2\}, ..., C_n, \sigma(C_i \cup C_j)}{C_1, ..., C_i \cup \{L_1\}, ..., C_j \cup \{L_2\}, ..., C_n}\ \textit{resolution}$$

with σ *a m.g.u. of* L_1 *and* $\overline{L_2}$.

$$\frac{C_1, ..., C_i \cup \{L_1, ..., L_m\}, ..., C_n, \sigma(C_i \cup \{L_1\})}{C_1, ..., C_i \cup \{L_1, ..., L_m\}, ..., C_n}\ \textit{factorization}$$

with σ *a m.g.u. of* $L_1 \dots L_m$.

- a resolution proof for a set of clauses S is a derivation of S in the resolution calculus; the substitution σ is local for every rule application; variables in every clause C can be renamed

First-Order Resolution Calculus – Example

1. $\neg p(x), q(x), r(x, f(x))$
2. $\neg p(x), q(x), r'(f(x))$
3. $p'(a)$
4. $p(a)$
5. $\neg r(a, y), p'(y)$
6. $\neg p'(x), \neg q(x)$
7. $\neg p'(x), \neg r'(x)$
8. $\neg q(a)$ — from 3 and 6 with $[x \backslash a]$
9. $\neg r'(a)$ — from 3 and 7 with $[x \backslash a]$
10. $q(a), r(a, f(a))$ — from 1 and 4 with $[x \backslash a]$
11. $q(a), r'(f(a))$ — from 2 and 4 with $[x \backslash a]$
12. $r(a, f(a))$ — from 10 and 8 with $[x \backslash a]$
13. $r'(f(a))$ — from 11 and 8 with $[x \backslash a]$
14. $p'(f(a))$ — from 12 and 5 with $[y \backslash f(a)]$
15. $\neg p'(f(a))$ — from 13 and 7 with $[x \backslash f(a)]$
16. $\square$ — from 14 and 15

The Necessity of Factoring

(1) :	$p(u)$	$\vee$	$p(f(u))$
(2) :	$\neg p(v)$	$\vee$	$p(f(w))$
(3) :	$\neg p(x)$	$\vee$	$\neg p(f(x))$

A possible resolution derivation:

(4) :	$p(u) \vee p(f(w))$	by resolving (1) and (2), with $v = f(u)$
(5) :	$p(f(w))$	by factoring (4), with $u = f(w)$
(6) :	$\neg p(f(f(w')))$	by resolving (5) and (3), with $w = w', x = f(w')$
(7) :	$\square$	by resolving (5) and (6), with $w = f(w')$

Outline

- Reminder: Clausal Form Translations
- Reminder: Propositional Resolution
- Reminder: Unification
- First-Order Resolution
- **Soundness and Completeness**
- Compactness
- Summary

Soundness and Completeness

Theorem 5.1 (Soundness and Completeness of Resolution).

The resolution calculus is sound and complete, i.e.

- *if A is provable in the resolution calculus, then A is valid (if $\vdash A$ then $\models A$)*
- *if A is valid, then A is provable in the resolution calculus (if $\models A$ then $\vdash A$)*

Proof.

See Ben-Ari, section 10.5, [Robinson 1965]. □

Soundness

Definition 5.1.

An interpretation $\mathcal{I}$ satisfies a clause C if for every variable assignment α, there is a $L \in C$ with $v_{\mathcal{I}}(\alpha, L) = T$.

So $\mathcal{I} \models \{p(x), q(x)\}$ if either p or q holds for all domain elements.

Lemma 5.1.

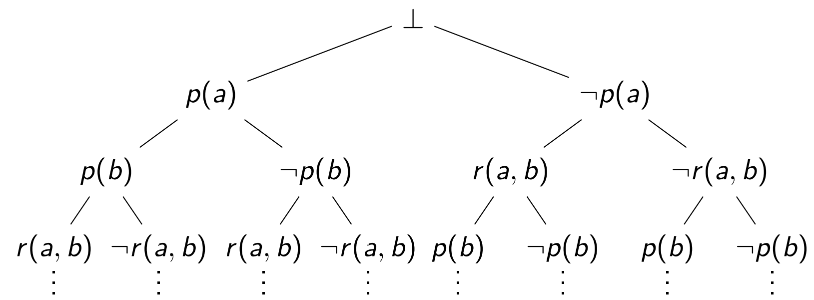
If a set of clauses S is satisfiable, then the result of adding the resolvent of two clauses $C_1, C_2 \in A$ to S is also satisfiable.

Proof.

Sketch: if $\mathcal{I} \models C_1$ and $\mathcal{I} \models C_2$ then also $\mathcal{I} \models \sigma(C_1)$ and $\mathcal{I} \models \sigma(C_2)$ (where σ is the m.g.u.) due to the substitution lemma.
Then $\mathcal{I} \models \sigma((C_1 \setminus \{L_1\}) \cup (C_2 \setminus \{\overline{L_2}\}))$ like for propositional logic. □

Completeness

- Semantic Trees can be infininte
- Define complete semantic trees for all closed literals



- Same notions of failure nodes and closed semantic trees as before
- There are resolution steps from *closed instances* of clauses
- Lifting: There are corresponding steps using m.g.u.s

Outline

Compactness

Observation

Nowhere in the definition of resolution do we need that S is finite.

- If S is unsatisfiable there is a closed semantic tree which enables a resolution step that gives a smaller semantic tree.
- No need to use *all* of S
- The closed tree is always finite (König's Lemma)
- To close the semantic tree we need only finitely many clauses $S' \subseteq S$.
- Collect all clauses $S_0 \subseteq S$ that are used in a refutation
- $S_0 \subseteq S$ is finite and unsatisfiable

Theorem 6.1 (Compactness).

Every unsatisfiable set of clauses S has a finite unsatisfiable subset

Compactness: Example

$$\exists x \neg p(x),\ p(a),\ p(fa),\ p(ffa),\ p(fffa),\ \ldots$$

- Every finite subset is satisfiable.
- E.g. take a domain with an extra element $d \in D$ that is not the value of any $f^n(a)$
- Interpret p such that $p^\iota(d) = F$, and therefore $\mathcal{I} \models \exists x \neg p(x)$.
- By compactness: The whole set is also satisfiable

Compactness: Counterexample

- Now we look at satisfiability 'over $\mathbb{N}$'
- i.e. in interpretations with $D = \mathbb{N}$, $0^\iota = 0$, $1^\iota = 1, \ldots$

$$\exists x \neg p(x),\ p(0),\ p(1),\ p(2),\ p(3),\ \ldots$$

- Every finite subset $S_0 \subseteq S$ is satisfiable over $\mathbb{N}$.
- E.g. let n be maximal with $p(n) \in S_0$
- Interpret $p(0) \ldots p(n)$ as true, but $p(n+1)$ as false.
- Then all $p(\cdots) \in S_0$ are satisfied and also $\exists x \neg p(x)$.
- But the whole set of formulas is unsatisfiable over $\mathbb{N}$

Theorem 6.2.

Satisfiability over the natural numbers is not compact.

Reasoning about numbers involves more than just first-order logic.

Outline

- Reminder: Clausal Form Translations
- Reminder: Propositional Resolution
- Reminder: Unification
- First-Order Resolution
- Soundness and Completeness
- Compactness
- Summary

Summary

- resolution calculus is one of the most popular proof search calculi for (classical) first-order logic
- consists of:
  - one axiom
  - resolution rule
  - factorization rule
- unification is used to unify terms of complementary literals
- easy to implement, but for an efficient proof search the application of the resolution rule needs to be controlled
- implemented in popular automated theorem provers, e.g. Otter, Prover9, Vampire
- Compactness: we can reason over (countably) infinite clause sets, but 1st-order reasoning is not strong enough for all of maths
- Next Week: logic programming and Prolog